Пояснювальна записка
до дипломного проекту
Войтенка Артема Віталійовича
на тему: «Інтегрований веб-застосунок для допомоги у повсякденному житті»

Київ – 2019 рік

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

SPA — Single Page Application;

SPI — Single Page Interface;

HTML — HyperText Markup Language;

CSS — Cascading Style Sheets;

ПЗ — Програмне забезпечення;

URI — Uniform Resource Identifier;

URL — Uniform Resource Locator;

HTTP — HyperText Transfer Protocol;

API — Application programming interface;

UX — User Experience;

UI — User Interface;

JSX — JavaScript XML;

DOM — Document Object Model;

СУБД — система управління базами даних;

JSON — JavaScript object notation;

SDK — Software Development Kit;

ВСТУП

У сучасному житті для того щоб бути конкурентоспроможною, людині необхідно тримати під контролем багато сфер життя з яких постійно надходить інформація, а також бути самоорганізованою. Зокрема, важливим аспектом самоорганізації являється планування часу та задач, контроль фінансів та збереження важливих записів. Але усю цю інформацію досить важко тримати у голові.

Для таких цілей дуже велика кількість людей до сьогоднішнього дня використовує різні блокноти, щоденники та записники. Але усе це має велику кількість недоліків, яких легко можна позбавитись завдяки використанню сучасних технологій.

За останні десятиріччя дуже поширилось використання різних пристроїв, що розширюють людські можливості, зокрема смартфони, комп'ютери, планшети, смарт-годинники. У 2019 році майже кожен має смартфон, який дозволяє виконувати усі функції блокноту , але з дуже розширеними можливостями і великою кількістю переваг. По-перше інформацію у блокноті дуже складно структурувати. Наприклад, необхідно знайти певний запис у блокноті, але людина точно не пам'ятає на якій сторінці він знаходиться, з використанням пристрою проблема вирішується реалізацією пошуку по записам. Або, наприклад, людині необхідно здійснювати контроль своїх витрат. Підрахунок і структуризація записів, а також отримання інформації за певні періоди стає великою проблемою для користувачів блокноту. Це завдання становиться дуже простим з використанням спеціального програмного продукту, який робить усе це автоматично. По-друге, блокнот у якому зберігається велика кількість важливої інформації легко забути або втратити, крім того не завжди зручно носити його з собою. Відновлення втраченої інформації являється неможливим У випадку використання програмного продукту, цих проблем дуже просто уникнути, завдяки розвитку інтернет технологій. Підключення до

інтернету дає нам можливість отримати усю цю інформацію майже з будь-якого пристрою у будь-якому куточку світу.

Переваги і доцільність використання саме програмного продукту очевидні. І в наш час існує досить велика кількість застосунків схожого характеру, але одним з основних їх недоліків є те що, наприклад, для контролю фінансів існує один застосунок, для створення списків задач — інший, а для зберігання ідей та важливих текстових записів — третій. Тому люди продовжують використовувати блокноти, адже вони дозволяють зберігати усе це в одному місці, що являється головним аспектом зручності. Більшість програмних продуктів не відповідають усім умовам універсальності та зручності. Вирішення наведених вище проблем, може значно полегшити людям організацію контролю свого повсякденного життя.

Отже метою дипломного проекту є реалізація веб-застосунку, який поєднав би у собі найбільш важливі функції, для контролю важливих сфер повсякденного життя, таких як: фінансовий облік, збереження важливої текстової інформації, облік задач, і, на додачу, як додатковий функціонал, яким зручно скористатися знаходячись на сторінці застосунку — перегляд погодних умов.

1 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

На даний момент існує велика кількість програмних застосунків для вирішення питань фінансового контролю, збереження текстової інформацій, та програм — таск-менеджерів, але рішення, яке б об’єднувало усі ці можливості в єдиний кінцевий продукт на сьогоднішній день не існує. Тому розглянемо найбільш популярні рішення по кожній з підпрограм окремо.

Money Pro — умовно безкоштовний застосунок, що призначений для контролю витрат та доходів. Має досить широкий спектр можливостей для фінансового обліку, таких як: формування звітів у текстовому вигляді та вигляді діаграм, нагадування про здійснення фінансових операцій, фільтрація операцій, створення бюджетів, моніторинг прогресу, індикація перевитрат, можливість інтернет-банкінгу (дає можливість підключатися до банку та скачувати банківські операції), планування регулярних операцій , калькулятор валют, наявність великої кількості опцій та налаштувань.

Переваги застосунку Money Pro:

- різноманітність функцій та можливостей;
- чудово продумана структура проекту;
- кросплатформність ;
- наявність можливості змінювати дизайн;

Недоліки застосунку Money Pro:

- широкий спектр можливостей надається тільки за умови платної підписки;
- перевантаженість функціональністю, що ускладнює зручність користування;
- відсутність можливості користуватися застосунком у браузері, що викликає проблеми користування на будь-якому пристрої;

Microsoft OneNote — програмний продукт для створення та зберігання заміток. Надає користувачам достатньо широкі можливості по управлінню

текстовою інформацією, серед яких: можливість групування заміток по тегам, додавання вкладень, копіювання веб-сторінок, швидкий пошук потрібних записів, редагування заміток, сканування документу, можливість створення рукописних заміток.

Переваги застосунку:

- підтримка мультимедійних заміток;
- надійність;
- зручний інтерфейс для користувачів Office;

Недоліки:

- повільність та незграбність;
- необхідність OneDrive для деяких можливостей;
- небезкоштовний;
- обмежений пошук у веб-версії;

Todoist — досить потужний застосунок для полегшення самоорганізації, який дозволяє створювати різні задачі, чек-лісти, зручно групувати їх та редагувати. Може синхронізуватися з іншими пристроями. Також дозволяє створювати шаблони проектів, які схожі між собою, для повторного використання.

Переваги:

- повна кросплатформеність, враховуючи браузерну версію;
- зручний та зрозумілий користувацький інтерфейс;
- можливість створення резервних копій даних;

Недоліки:

- багато корисних функцій надаються після покупки платної підписки;
- деякі незрозумілі на перший погляд кнопки та іконки;
- відсутність можливості створення нагадувань за допомогою розпізнавання людської мови;

Проаналізувавши існуючі рішення та їх можливості, переваги та недоліки, можна висунути такі критерії, на які треба зважити у процесі розробки власного продукту:

- застосунок має працювати на будь якому пристрої без встановлення будь якого додаткового програмного забезпечення;
- застосунок не має бути перегруженим зайвим функціоналом;
- інтерфейс має бути простим та інтуїтивно зрозумілим;
- варто приділити увагу красивому візуальному оформленню, що відповідає сучасним канонам веб дизайну;

Висновки до розділу 1

Отже, був виконаний аналіз предметної області та існуючих рішень, на основі якого можна визначити напрямок подальшої розробки власного застосунку. У розділі описано деякі вимоги до розроблюваного програмного продукту, а також виділено аспекти, на які варто звернути увагу при плануванні та розробці застосунку.

2 ВИБІР ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ

2.1 Вибір мови розмітки та стилів

Мова розмітки тексту в комп'ютерній термінології — набір символів або послідовностей, що вставляються в текст для передачі інформації про його виведення або будову. Текстовий документ, написаний з використанням мови розмітки, містить не тільки сам текст (як послідовність слів і розділових знаків), а й додаткову інформацію про різні його ділянки — наприклад, вказівку на заголовки, виділення, списки і таке інше. У більш складних випадках мова розмітки дозволяє вставляти в документ інтерактивні елементи і зміст інших документів.

2.1.1 Вибір мови розмітки

Стандартом у сучасній світовій павутині являється мова розмітки HTML, що в перекладі на українську означає «мова гіпертекстової розмітки». Загалом вона відповідає за розташування у документі текстів, зображень, мультимедійних файлів або таблиць. Основою мови являються HTML теги, кожен з яких несе в собі певну інформацію, яка описує .вміст веб-сторінки.

Для зручного оформлення зовнішнього вигляду сторінки у проекті використовуються принципи блочної верстки, з активним використанням тегу . Верстка базується на засадах і можливостях сучасного стандарту мови розмітки — HTML5 [1].

2.1.2 Вибір мови для стилізації

CSS — формальна мова, що слугує для стилізації зовнішнього вигляду документу, що написаний за допомогою мови розмітки. Розробники веб-сторінок використовують дану мову для задання зовнішнього та внутрішнього

позиціонування блоків, розмірів блоків, текстів, зображень, задання кольорів та інше.

Варто зауважити, що сучасні Front-end розробники використовують для зручності різні CSS препроцесори, які значно спрощують написання стилів для елементів документу. Це так званий синтаксичний цукор для розробників, який використовує власний, зручний синтаксис, котрий компілюється у звичайний CSS.

Серед найбільш корисних можливостей препроцесорів можна виділити:

- маніпуляції з кольорами;
- можливість створювати міксини, які автоматизують певні процеси;
- можливість імпорту файлів;
- можливість проведення математичних операцій;
- можливість використання синтаксису вкладеності, що значно спрощує і скорочує написання коду;

Серед найбільш відомих та популярних препроцесорів можна виділити декілька основних, таких як: SASS, LESS, Sylus, Haml. Для реалізації візуального оформлення даного проекту використовується препроцесор SASS, так як він має більш зручний синтаксис, підтримує і чистий CSS; а також має більш широкий спектр можливостей у порівнянні з іншими.

Також у сучасній Front-end розробці, для зручності підтримки, масштабування і перевикористання певних блоків використовують CSS-методології, що основані на деяких загальноприйнятих для конкретної методології правилах. Серед найпопулярніших можна виділити такі, як: BEM, SMACSS, ECSS [2]. Однак найбільш ефективним та розповсюдженим являється БЕМ методологія, яка власне і використовується на проекті.

БЕМ — розшифровується як «Блок, Елемент, Модифікатор». В її основі лежить принцип розподілення інтерфейсу на незалежні блоки. Методологія дозволяє легко і швидко розроблювати інтерфейси будь-якого рівня складності і повторно використовувати існуючий код, що допомагає запобігти дублюванню.

Головні принципи BEM методології:

- не використовувати ідентифікатори, оскільки це заважає повторно використовувати код;
- не використовувати селектори тегів, так як HTML розмітка сторінок нестабільна, і при зміні тегу одразу зламаються стилі;
- не використовувати універсальний селектор, оскільки даний селектор задає загальний стиль для проекту, що впливає на усі вузли при верстці. Це накладає певні обмеження на повторне використання верстки в іншому проекті;
- не використовувати вкладені селектори, бо такі селектори збільшують зв’язність коду і ускладнюють повторне використання
- не використовувати комбіновані селектори, так як комбіновані селектори збільшують специфічність і тому перевизначати блоки стає набагато складнішим;
- не суміщати клас і тег у селекторі, як і у випадку з комбінованими селекторами, збільшується специфічність;
- не використовувати селектори атрибутів, оскільки вони є менш інформативними, аніж селектори по класам;

Отже використання даної методології є дуже важливим фактором, який впливатиме на якість розробки, а також підтримки розроблюваного продукту.

2.2 Вибір мови програмування

Java-Script — прототипно-орієнтнована скриптова мова програмування. Проте мова підтримує не тільки прототипну парадигму програмування, а й інші, такі як: декларативна, функціональна та імперативна. Раніше мова використовувалася для розробки клієнтської частини застосунків, як інструмент взаємодії з користувачем, управління поведінкою браузеру, обміну даними з

сервером, зміни візуального відображення сторінки та інше. З появою платформи Node.js вона стала універсальною, тобто відтепер мова може використовуватися і для реалізації серверної частини програмного продукту. Сьогодні, JavaScript можна сміливо назвати однією з найпопулярніших мов програмування, оскільки його можливості майже необмежені. Його можна використовувати на більшості мобільних платформ, у тому числі для розробки застосунків на операційні системи Android та iOS.

При розробці застосунку використовуються сучасний стандарт ECMAScript 2015, або ж ES6 [11]. У цьому стандарті включена велика кількість покращень та спрощень для більш зручної розробки.

2.3 Огляд та вибір допоміжних JavaScript інструментів

На сьогоднішній день існує велика кількість різних JavaScript бібліотек та фреймворків, призначених для такої важливої та складної задачі, як синхронізація користувацького інтерфейсу і внутрішнього стану застосунку. Для реалізації подібних інтерфейсів, що потребують синхронізації, на чистому JavaScript без використання допоміжних інструментів потрібен у багато разів більший об'єм коду та часових затрат. Крім цих проблем, постає також питання ефективності, та швидкодії. Головною ж проблемою являється необхідність оновлення користувацього інтерфейсу при кожній зміні стану застосунку. При кожному оновленні, необхідно докласти неабияких зусиль, що виражаються відповідним кодом. Такий код не тільки важко писати й підтримувати, він також є недостатньо надійним, і його достатньо легко «зламати».

Отже використання інструменту для оновлення станів є практично обов'язковою ознакою сучасної Front-end розробки.

Серед найпопулярніших JavaScript інструментів можна виділити бібліотеку JQuery, React.js, а також фреймворки Angular.js та Vue.js.

2.3.1 Стратегії синхронізації інтерфейсу та стану

Існує дві основних стратегії синхронізації інтерфейсу та стану:

- повторний рендеринг усього компоненту. Так працює бібліотека React.js. Її суть заключається в тому, що коли стан компоненту змінюється, бібліотека рендерить DOM у пам’яті і порівнює його з існуючим на сторінці. Так як робота з DOM досить ресурсоємна операція, бібліотека генерує те, що називається Virtual DOM, і результат порівнює з минулою версією Virtual DOM, після чого бібліотека знаходить відмінності і вносить зміни у DOM сторінки. Цей процес називають погодженням з англійської reconciliation;
- відсліджування змін з використанням наглядачів. Так працюють Angular.js та Vue.js. Змінні стану є об’єктами за якими здійснюється нагляд, і коли вони змінюються, оновлюються лише ті DOM-елементи, на котрі впливають ці зміни, що призводять до оновлення інтерфейсу;

2.3.2 Порівняння основних JavaScript фреймворків та бібліотек

JQuery — невелика, але багата можливостями, написана на мові JavaScript . Якщо говорити простіше, jQuery - це набір корисних функцій, що спрощує маніпулювання елементами HTML і стилями CSS, створення AJAX-запитів, обробку подій, наприклад, клік миші, управління анімацією — словом, того, що називається «інтерактивність», взаємодія з користувачами сайту .

Переваги JQuery:

- кросбраузерність. Синтаксис JQuery підтримується усіма веб-браузерами, і можна бути впевненим, що кожен користувач зможе побачити і скористатися написаним на цій бібліотеці плагіном;

– компактність коду. Те що на чистому JavaScript доводиться писати в окремих функціях, на JQuery реалізовується в декілька рядків коду;
– зручна робота з подіями та візуальними ефектами;
– зрозуміла документація. На офіційному росойськомоввному ресурсі можна знайти і ознайомитися з усіма можливостями та функціями бібліотеки, що зручно розбиті на категорії;
– на JQuery створено тисячі зручних плагінів. Для того щоб заощадити ресурси, можна скористатися одним з готових них, наприклад фотогалерея або плагін для валідації форм;

Недоліки JQuery:

– професіонали стверджують, що код, написаний на чистому JavaScript працює швидше. Але на сьогоднішній день перевірити це майже неможливо, оскільки розробники бібліотеки постійно пришвидшують її швидкодію, а також з кожним роком потужність машин та швидкість інтернету постійно зростають, що дає можливість знехтувати різницею у швидкодії;
– розмір бібліотеки. Бібліотека JQuery важить приблизно 19кб, що теоретично може сказатися на швидкості завантаження сторінки, особливо на старих пристроях та нестабільному підключенні до інтернету;
– обмежений функціонал у порівнянні з більш сучасними бібліотеками та фреймворками;

Angular.js — фреймворк з відкритим вихідним кодом для створення Front-end застосунків від компанії Google. Його головною направленістю являється вирішення задач з якими стикається розробник при побудові односторінкових застосунків. Він реалізує підходи Model-View-Controller та Model-View-ViewModel. Фреймворк працює з HTML, що містить додаткові атрибути, які описуються директивами, і пов'язує введення або виведення області сторінки з моделлю, яка представляє собою звичайні змінні JavaScript. Значення цих змінних

задаються вручну або витягуються з статичних або динамічних JSON-даних. Поєднує у собі двостороннє зв’язування даних, декларативні шаблони, впровадження залежності.

Переваги Angular.js:

- MVVM модуль дає можливість окремо оперувати в одному розділі програми, використовуючи той же набір даних;
- стабільна та довгострокова підтримка Google;
- взаємозв’язок функцій;
- дуже швидка компіляція;
- двостороння прив’язка даних, що мінімізує ризики виникнення помилок;

Недоліки Angular.js:

- додатки створені за допомогою рамкової системи JavaScript, надмірно розряджають акумулятор пристрою;
- застосунки на основі Angular js вимагають великої оптимізації для рішення проблем з низькою продуктивністю;
- високий поріг входження для освоєння;
- при переході з однієї версії на іншу можуть виникати багато інтеграційних помилок;

Vue.js — відносно молодий фреймвок, що поєднує у собі Angular.js та React.js. Його ще називають прогресивним JavaScript фреймворком. Основна сфера застосування це створення та організація клієнтського інтерфейсу. Основними концепціями являються конструктор, компоненти, директиви та переходи. На відміну від фреймворків-монолітів, Vue створений придатним для поступового впровадження. Його ядро в першу чергу вирішує завдання рівня уявлення (view), що спрощує інтеграцію з іншими бібліотеками та існуючими проектами. З іншого боку, Vue повністю підходить і для створення складних односторінкових застосунків Single-Page Applications, якщо використовувати його спільно з сучасними інструментами та додатковими бібліотеками.

Переваги Vue.js:

- докладна документація, що може прискорити освоєння фреймворку;
- адаптивність, забезпечує відносно швидкий перехід від інших фреймворків завдяки своїй спорідненості з Angular та React з точки розу дизайну та архітектури;
- чудова інтеграція, дозволяє легко інтегрувати невеликі інтерактивні частини в одну структуру;
- маленький розмір, всього 20 кб, при цьому фреймворк зберігає сврою швидкість там гнучкість, що дозволяє досягнути досить високої продуктивності;

Недоліки Vue.js:

- через те, що Vue достатньо молодий, він все ще має достатньо низьке поширення у порівнянні з іншими фреймворками;
- менша підтримка Communiti, має досить невеликий обсяг матеріалів для вивчення;
- ризик надмірної гнучкості;
- робота над станом застосунку відбувається "під капотом". У масштабних проектах це може вилитися в неочевидність роботи компонентів, що часто створює проблеми з налагодженням і ефективної розробкою;
- компонентний підхід у Vue.JS не такий гнучкий і очевидний, як в React;
- система рендеринга React більш функціональна. Вона надає більше можливостей для налагодження;
- шаблонізації React значно гнучкіша (JS vs DSL);

React.js — бібліотека від Facebook, яка являється домінуючою серед засобів для побудови SPA [3]. Створена для вирішення питання з частковим оновленням вмісту сторінки при розробці SPA. Ідеально підходить як для створення великих веб-застосунків так і простих застосунків. React володіє власним віртуальним

DOM, який управляє фактичним DOM браузера і так як він набагато швидше, ніж DOM браузера, він значно підвищує продуктивність. DOM React може створювати більше 200 000 вузлів в секунду, що перевищує середній показник вузлів для більшості сайтів. DOM може відтворювати зміни завдяки використанню алгоритму Diffing, який здатний скоротити обчислення різниці від складності з n^3 до n. При роботі надає реактивність, основується на компонентах, фокусується на кореневій бібліотеці, виносячи такі питання як роутинг або управління глобальним станом у додаткові бібліотеки.

Переваги:

- у нього є потужна спільнота;
- для нього розроблено безліч сторонніх бібліотек, які допомагають вирішувати різні завдання;
- існують корисні додаткові компоненти для цього фреймворку;
- є розширення для браузерів, які допомагають налагоджувати застосунки, створені за допомогою даної бібліотеки;
- безліч документації та онлайн-ресурсів. Завдяки підтримці Facebook існує безліч можливостей використовувати документацію та онлайн-ресурси для навчання та використання React;
- високий рівень відклику та гнучкості;
- віртуальна DOM структура [4], яка дозволяє впорядковувати документи форматів HTML, XHTML або XML в дерево, яке найкраще підходить веб-браузерам для аналізу різних елементів веб-застосунку;
- функціонал React сприяє покращенню роботи UI;

Недоліки:

- необхідні засоби для зборки;
- через те, що бібліотека дуже стрімко розвивається, документація розміщується трохи хаотично;
- вищий поріг входження у порівнянні з Vue.js;
- не зовсім точний алгоритм Virtual DOM;

– при спілкуванні з сервером необхідне складне асинхронне програмування;

Також варто зауважити, що React.js має власне розширення синтаксису JavaScript, що нагадує XML, і дозволяє використовувати синтаксис HTML тегів для рендерингу компонентів. Код написаний з використанням JSX компілюється у виклики методів бібліотеки React. На практиці це дуже зручний інструмент, що надає бібліотеці велику перевагу серед інших.

При аналізі усіх недоліків та переваг найбільш популярних фреймворків [6] для побудови SPA було прийнято рішення використовувати React.js, оскільки його концептуальні засади та архітектура найбільш відповідають вимогам даного проекту.

2.4 Огляд технологій, що супроводжують розробку на бібліотеці React.js

Так як обрана бібліотека відповідає за View у шаблоні MVC, одним з рішень для управління Model можна обрати Redux. Redux — інструмент для управління станом даних та станом інтерфейсу в JavaScript застосунках. Він підходить для SPA в яких управління станом з часом може ставати складним та заплутаним.

В наш час для полегшення відладки та контролю розробки для більшості популярних технологій існують спеціальні браузерні розширення. Для спрощення користуванням Redux, відладки та відсліджування зміни станів застосунку існує розширення для браузеру Google Chrome Readux DevTools. Воно дозволяє дуже розгорнуто оглянути усі дії, що пов’язані з обміном даними, зміною властивостей компонентів та їх станів.

Також існує браузерне розширення React Developer Tools, яке доступне у Google Chrome, що дозволяє легко перевіряти ієрархію компонентів в інструментах розробника (з англ. — Chrome Developer Tools). Відповідно можна дослідити кореневі елементи, які були відрендерені на сторінці, а також дочірні компоненти. Вибравши один з компонентів у дереві, можна перевірити його

поточні властивості та стан на панелі з правого боку. Переходячи по хлібним крихтам можна проінспектувати обраний компонент, компонент, що створив його, і так далі по ієрархії.

2.5 Налаштування робочого середовища та підбір інструментів розробки

Налаштування робочого середовища це обов'язковий етап у розробці будь-якого застосунку. Сучасні застосунки, написані на JavaScript мають досить розгалуджену структуру ресурсів та модулів, яка постійно ускладнюється, тому необхідно використовувати застосунок, що здійснює зборку застосунку. На даний момент найпотужнішим засобом для цього являється Webpack. Від дозволяє вирішувати дуже велику кількість задач. Webpack дозволяє скомпілювати усі JavaScript модулі в єдиний файл, транспілювати sass код у звичайний css, мініфікувати CSS та JS код, може конвертувати та мініфікувати зображення, може розділити вихідний файл на декілька файлів для запобіганню повільного завантаження сторінки через надмірний розмір JS файлу. За умови використання Babel можна використовувати найновітніші можливості останніх стандартів ECMAScript без побоювання за підтримку браузерами, оскільки Webpack у зв'язці з Babel транспілює сучасний код у більш старі версії.

Важливим моментом у розробці є дотримання єдиних домовленостей у організації структури проекту, і для кожного стеку певних технологій є свій набір вимог. Так як ці вимоги у всіх проектах з однаковими технологіями схожі, для спрощення організації і більш швидкого старту безпосередньої розробки завдяки позбавлення від рутинної роботи використовується деякий шаблон-генератор. У програмуванні це має назву скаффолдинг. І так як для розробки було обрано фреймворк React, можна скористатися інструментом для швидкого старту проекту "create-react-app".

Для встановлення усіх необхідних модулів та залежностей використовується пакетний менеджер npm, що базується на платформі Node.js,

про яку вже згадувалось раніше. Саме завдяки йому можна дуже легко підготувати і налаштувати середовище.

Висновки до розділу 2

У розділі було розглянуто та проаналізовано найпопулярніші технології, які підходять для реалізації поставлених завдань, а також допоміжні інструменти, що спрощують виконання тих чи інших задач. Отже, обраний стек технологій для виконання дипломного проекту складається з мови для розмітки документу HTML, мови для стилізації документу CSS для зручності роботи з якої використовується препроцесорна мова SASS, JavaScript як основна мова програмування у зв’язці з бібліотекою React, яка в свою чергу використовує допоміжну бібліотеку Redux для керування властивостями та станами застосунку. Збірка рішення виконується інструментом Webpack.

3 ФОРМУВАННЯ ВИМОГ ДО ІНТЕРФЕЙСУ ТА ФУНКЦІОНАЛУ

3.1 Формування вимог до інтерфейсу

Головною частиною розроблюваного застосунку є саме веб інтерфейс. З одного боку використання мобільних або десктопних застосунків було б в деякій мірі зручнішим, але з іншого завдяки використанню веб браузеру, при розробці можна не залежати від пристрою.

UI — це вид інтерфейсів, представниками в якому з однієї сторони виступає людина (користувач), а з іншої машина та пристрій. Такий інтерфейс представляє набір методів та засобів для взаємодії користувача зі складними пристроями, машинами та апаратурою. Він являється однією з перших сходинок при розробці програмного продукту і слугує для того щоб результуючий застосунок був інтуїтивно зрозумілим, привабливим та максимально простим у використанні за допомогою деяких графічних рішень. Він складається з двох складових: обладнання і програмне забезпечення. На сьогоднішній день можна виділити деякі обов’язкові правила UI дизайну:

- організованість елементів інтерфейсу. Це означає, що вони мають бути логічно згруповані та взаємопов'язані.
- групування елементів інтерфейсу. Має на увазі під собою об’єднання в логічні групи логічно зв’язаних елементів.
- вирівнювання елементів інтерфейсу. Стильове оформлення відіграє неабияку роль, адже воно залишає перше враження у пам’яті користувача.
- єдиний стиль елементів інтерфейсу.
- наявність вільного простору. Це дозволяє розмежовувати інформаційні блоки фіксуючи увагу на чомусь одному.

User Experience Design, що в перекладі означає досвід взаємодії включає у себе різні UX-компоненти, а саме: інформаційну архітектуру, проектування взаємодії, графічний дизайн та контент.

В цілому, UX дизайн являє собою комплексний підхід до взаємодії користувача з інтерфейсом, не важливо чи то веб-сайт, мобільний застосунок або будь-яка інша програма. При розробці дуже важливо максимально врахувати усі тонкощі, починаючи від середи користувача і закінчуючи способами вводу та відображення інформації. Основні питання, які вирішує UX дизайн:

- постановка цілей і задач — чого в результаті треба досягнути;
- підбір відповідних інструментів для реалізації цілей;
- розробка продукту, що максимально зручний та зрозумілий для сприйняття цільовою аудиторією;
- аналіз кінцевого результату — чи відповідає продукт сподіванням і наскільки високий рівень задоволеності користувачів;

Окремо варто відмітити IA, що розшифровується як «Information architecture» і в перекладі означає — інформаційна архітектура. Її діяльність сфокусована на організації даних, тобто наскільки інформація являється структурованою з точки зору користувача, а не технічних або системних правил. Вона визначає розміщення елементів на сторінці, зв’язок самих сторінок.

3.2 Формування вимог до функціоналу

Ідея розроблюваного застосунку полягає в об’єднанні декількох підпрограм в єдиний результуючий продукт. Цей продукт має складатися з підпрограм, кожна з яких буде являтися окремою сторінкою, і усі вони будуть поєднані між собою наскрізьним головним меню, яке дозволить швидко перемикатися з однієї підпрограми до іншої та головною сторінкою, що буде супроводжувати користувача при вході на сторінку застосунку. Складовими частинами кінцевого продукту є такі підпрограми:

- контроль фінансів;
- записник;
- таск-менеджер;

– застосунок для швидкого перегляду погодних умов;

Варто чітко сформувати вимоги до функціоналу кожної з підпрограм для адекватної оцінки трудових та часових затрат на реалізацію кінцевого продукту, а також формування приблизного графіку проведення роботи над кожною функціональною частиною.

3.2.1 Вимоги до функціоналу підпрограми фінансового контролю

Головною ідеєю цієї частини проекту є на перший погляд примітивний і легкий у користуванні засіб для контролю власного бюджету, але який в свою чергу надасть широкі можливості користувачу в аналізі своїх записів.

Функції підпрограми, які очікуються у кінцевому результаті її реалізації:

– користувач може додавати записи про здійснені фінансові операції, які передбачають в собі додавання назви операції, її суми, інформаціїї про те чи ця операція являється надходженням чи витратою, сфери, до якої належить фінансова операція, розширеної інформації про цю, що являє собою текстові дані та зображення, а також передбачається вибір дати;
– на сторінці має відображатися секторна діаграма, яка за замовчуванням відображає відношення надходжень до розходів;
– передбачається фільтрація фінансових операцій, що дозволить користувачу докладно дослідити його фінансові дії;
– фільтрація за датою, що дає можливість обрати період;
– фільтрація за типом операції (усі, надходження, витрати);
– фільтрація за сферою до якої належить операція;
– усі типи фільтрів мають працювати у зв’язці один з одним;
– користувач може редагувати запис про операцію, з можливістю повної зміни введених даних;
– видалення записів;

– перегляд детальної інформації про той чи інший запис;
– відображення статистичних даних, що являють собою підрахунки, які базуються на сумарних значеннях грошових сум стосовно кожної сфери до якої належать фінансові операції;
– відображення статистичних даних щодо операцій у вигляді бульбашкової діаграми;

Як було вказано вище, усі типи фільтрів мають залежати один від одного. Іншими словами, якщо, наприклад обрати фільтр періоду з певного числа по інше, і в цей же час обрати фільтр сфер, то мають відобразитися тільки записи що знаходяться у межах цього періоду і належать тій сфері, яка обрана у фільтрі.

Щодо функціональності інтерфейсу і його реакції на дії користувача, можна виділити наступні пункти:

– при виборі типу операції (надходження чи витрата) автоматично має змінюватись список сфер в залежності від обраного типу;
– якщо обов’язкові поля не заповнені, користувач має бути сповіщений про помилку;
– при підтвердженні додавання нового запису, блок для створення має автоматично закритися, і відобразитися актуальний список фінансових операцій;
– при зміні фільтрації, усе що стосується зміни даних (список записів, діаграми, статистичні дані) мають змінюватися динамічно;
– при кліку на зоні, що відповідає за вибір дати, має відкриватися календар, який автоматично ховається при виборі;
– клік по іконці інформації на записі відображає детальний опис цього запису і ховає діаграму;
– клік по іконці редагування відкриває запис у блоці редагування
– клік по кнопці «Add item», відкриває форму для створення нового запису, ховаючи при цьому вже створені записи;

3.2.2 Вимоги до функціоналу підпрограми записника

Ідеєю цієї підпрограми є зберігання записів користувача, так званих «нотаток» у зручному вигляді, з деякими додатковими можливостями.

Серед умов до функціоналу можна виділати такі:

- додавання запису, що передбачає додавання назви, короткого опису, тегу, кольору та розширеного запису;
- редагування основного тексту запису через текстовий редактор, з можливістю стилізації кольору, розміру шрифту, сімейства шрифту, розташування тексту, використання різних видів списків і т. п.;
- видалення запису;
- редагування запису;
- швидкий пошук записів по тексту;
- сортуваня записів по тегам;

3.2.3 Вимоги до функціоналу підпрограми таск-менеджеру

Ідея цієї підпрограми полягає у швидкому створенні списків із завданнями, з можливістю розбиття їх на підзадачі, контролю статусу виконання цих завдань та їх групуванню.

Основні вимоги до функціональності:

- додавання нових завдань
- групування завдань на різні списки
- розподілення завдань по групам
- планування задач
- сортування завдань за стадією їх виконання
- архівування списків з виконаними завданнями

3.2.4 Вимоги до функціоналу підпрограми погоди

Завдання цієї підпрограми це відображення погодних умов в певному регіоні. Серед вимог можна виділити наступні:

- автоматичне визначення місцезнаходження користувача і відображення погодних умові відповідно до цього, при завантаженні сторінки цієї підпрограми;
- надання інформації стосовно погодних умов у регіоні, який запросив користувач;
- відображення максимальних і мінімальних температурних показників на декілька найближчих днів у вигляді графіку кривої;
- відображення різної статистичної інформації, такої як швидкість вітру, вологість, атмосферний тиск, індекс ультрафіолету і т. п.;

Спираючись на поставлені вимоги можна стверджувати, що для реалізації інтерфейсу найкращим рішенням буде створення односторінкового застосунку SPA. З візуальної сторони можна частково використати концепцію односторінкового інтерфейсу — Sinle Page Interface. Це такий інтерфейс, що вміщується на одній сторінці і за свою мету має наблизити користування веб-застосунком до настільної програми.

Головною суттю SPA є те, що сторінка не оновлюється при взаємодії користувача з нею, і навіть якщо користувач натиснув на посилання, зміна вигляду сторінки відбувається без її перезавантаження. При взаємодії з застосунком такого типу часто спостерігається динамічний зв’язок з веб-сервером.

Висновки до розділу 3

В даному розділі було розглянуто основні вимоги до інтерфейсу, а також функціоналу кожної з підпрограм застосунку. На основі цих вимогу базується уся розробка програмного продукту.

4 РЕАЛІЗАЦІЯ

4.1 Архітектура проекту

В основі роботи застосунку лежить так звана модель взаємодії клієнт-сервер. Серверна частина представлена у вигляді Firebase SDK [7, 16], що надає також і базу даних. А клієнтська сторона реалізована з використанням бібліотеки React.js та деяких допоміжних бібліотек, описаних нижче. Архітектура клієнтської частини має вигляд компонентної моделі, що дозволяє повторне використання коду, і за рахунок ізоляції різних модулів системи дає можливість знизити трудоємність і ресурсоємність розробки.

4.2 Архітектурний підхід Flux

Так як задуманий функціонал вимагає дуже тісної взаємодії з даними у багатьох компонентах та постійної зміни станів, при розробці серйозною складністю являється передача отриманої з бази даних інформації і подальша її передача до компонентів для обробки та використання у цих компонентах, а також передача станів та властивостей, що їх потребують компоненти. Тому доцільно буде використати одне місце для збереження цього всього, яке забезпечить доступ для всіх компонентів.

Оскільки застосунок працює з динамічними даними при реалізації проекту за основу було обрано архітектурний підхід Flux . Головною відрізняючою особливістю якого є одностороння направленість передачі даних між компонентами Flux [6] архітектури. Архітектура накладає обмеження на поток даних, зокрема виключаючи можливість оновлення стану компонентів самими собою. Такий підхід дає можливість контролювати потік даних, і зробити його максимально передбачуваним з точки зору виникнення можливих помилок та їх причин.

У своїй реалізації архітектура Flux складається з трьох прошарків, які взаємодіють по-порядку:

– actions (дії);
– stores (сховища);
– views (представлення);

Також варто відмітити, що посередником між діями та сховищем являється Dispatcher (диспетчер).

Головна суть цієї архітектури полягає в тому, що застосунок має одне велике дерево, в якому зберігається стан (state) застосунку — це згадане вище сховище (англ. store). Структура сховища представляє собою об’єкт з декількома рівнями вкладеності. В цьому об’єкті знаходиться усе, що стосується станів застосунку.

З метою втілити у життя вище описаний архітектурний підхід у розроблюваному застосунку, було прийнято рішення про використання бібліотеки Redux [10], яка надає повний функціонал для цього.

4.3 Опис застосування бібліотеки Redux

Для того щоб змінити стан у сховищі використовується метод Dispatch, який приймає Action creator, що в свою чергу повертає action type, який відображає вид виконаних дій. Також сюди включені дані, що описують дії. Reducer — функція , що приймає поточний стан та дію і повертає новий стан. При реалізації застосунку було використано лише один Reducer, якого цілком достатньо для трансформції станів та дій.

Для реалізації даного підходу використовується згаданий раніше інструмент Redux [17], загальну схему роботи якого відображено на Рисунку 4.2

Reducer організовано як чисту функцію («pure function»), тобто він не робить зовнішніх викликів по мережі чи базі даних, повертає значення, що залежить тільки від переданих параметрів. Аргументи цієї функціє являються

незмінними, тобто функції не можуть їх змінити. Виклик чистої функції з тими ж аргументами завжди повертає однаковий результат. На рисунку 4.1 візуалізується цикл обробки змін сховища із використанням Redux.

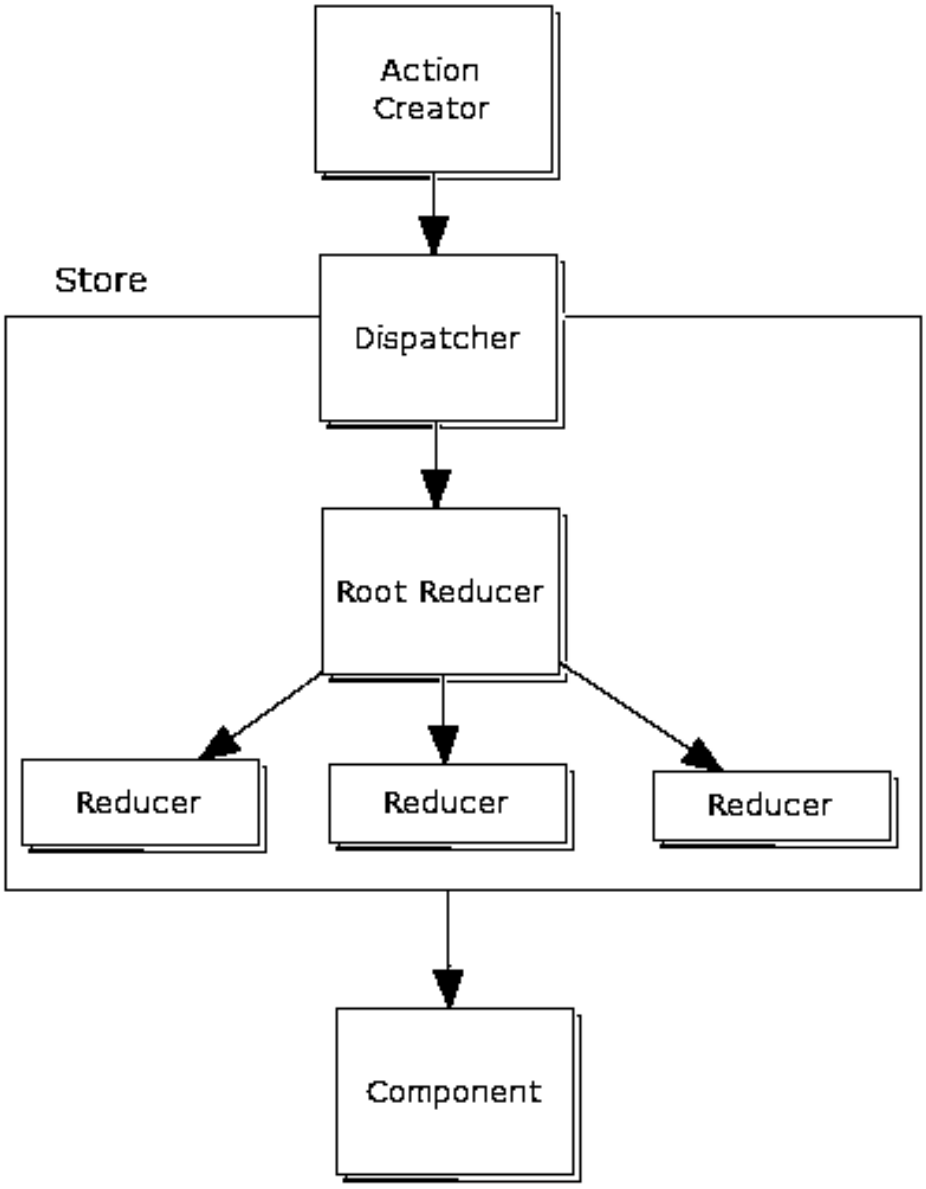

Рисунок 4.1 — загальна схема роботи Redux

Розглянемо цей підхід більш детально на прикладі реалізації підпрограми для контролю за фінансами. В наведеному на рисунку 4.2 лістингу, можна побачити код створення сховища за допомогою вбудованої функції бібліотеки Redux createStore(), аргументами до якої передаються reducer та спеціальні

властивості об’єкта window, що слугують для підключення засобів розробника Redux Dev Tools, які існують завдяки встановленню розширення для Google Chrome. Це розширення допомагає контролювати стан компонентів у сховищі, відсліджувати усі дії, що передаються до сховища, а також виконувати тестування правильної взаємодії компонентів зі сховищем.

```
const store = createStore(
  reducer,
  window.__REDUX_DEVTOOLS_EXTENSION__ && window.__REDUX_DEVTOOLS_EXTENSION__()
);
```

Рисунок 4.2 Створення «store»

Шматок коду, наведений на рисунку 4.3 представляє створення «Reducer». На ньому можна побачити обробку дії, яка називається «ON_ITEM_ADD» і відповідає за зміну даних при додаванні записів про фінансові операції. Ця дія відправляється з компоненту ItemCreator за допомогою «екшн креейтору» — «addItem». У випадку виконанання цієї дії, перезаписується стан сховища.

```
const reducer = (state = initialState, action) => {
  switch (action.type) {
    case "ON_ITEM_ADD": {
      return {
        ...state,
        data: action.payload
      };
    }
```

Рисунок 4.3 — створення «Reducer»

У наведеному на рисунку 4.4 лістингу можна побачити метод mapDispatchToProps, що слугує для виклику методу Dispatch з необхідним «Action» безпосередньо всередині класу компоненту, в якому можна отримати даний метод через властивості цього компоненту. Також для передачі стану сховища використовується службовий метод mapStateToProps.

```
const mapDispatchToProps = dispatch => {
  return {
    addItem: item => {
      dispatch(addItem(item));
    },
    changeCreatorVisibility: () => {
      dispatch(changeItemCreatorVisibility());
    }
  };
};
export default connect(
  mapStateToProps,
  mapDispatchToProps
)(InputGroup);
```

Рисунок 4.4 Підключення «Redux store» до компоненту

4.5 Вибір системи управління базами даних

Важливим кроком при розробці будь-якого застосунку являється вибір бази даних. База даних — організована структура. Що призначена для зміни та обробки взаємопов'язаної інформації, переважно великих об'ємів. Важливим аспектом є також вибір системи управління базами даних (СУБД). Для розв'язання різних задач існує багато різних СУБД, що основуються на моделі бази даних. Модель бази даних — це спеціальні структури, призначені для роботи з даними. Усі СУБД сильно відрізняються тим, у який спосіб вони зберігають і оброблюють свої дані. Найбільш популярною на сьогоднішній день являється реляційна система управління базами даних, хоча останнім часом дуже швидко набирають популярність нереляційні бази даних NoSQL. NoSQL спеціально створені для певних моделей даних і володіють гнучкими схемами. Що дає можливість розроблювати сучасні застосунки. Вони отримали широке розповсюдження у зв'язку із простотою розробки, функціональністю, швидкодії та будь-яких масштабах. У них застосовуються різноманітні моделі даних, у тому числі документні, графові, пошукові, з використанням пар «ключ-значення» і збереженням даних у пам'яті. У таблиці 4.1 наведено порівняльну характеристику SQL та NoSQL.

Таблиця 4.1 — порівняння типів СУБД

Тип СУБД	Реляційна	Нереляційна
Структури даних та їх типи	Використовуються строгі схеми даних	Допускають будь-який тип даних
Запити	Реляційні бази даних відповідають стандартам SQL, тому дані із них можна отримувати за допомогою мови SQL	Об'єктно-орієнтовані API дозволяють без складнощів здійснювати запис і витяг структур даних, розміщених в пам'яті. Завдяки використанню ключів секцій застосунки можуть вести пошук по парам «ключ-значення», наборам стовпців або частково структурованим документам, що містять серійні об'єкти і атрибути застосунків.
Підтримка	Мають дуже велику популярність, і велику історію, пропонують як платні так і безкоштовні рішення. При виникенні проблем є багато ресурсів, що можуть допомогти у їх вирішенні	Дуже стрімко набирають популярність, самі по собі являються більш простим рішенням

Продовження таблиці 4.1

Масштабованість	Легко піддається вертикальному масштабуванню (збільшенню системних ресурсів)	Легко піддається вертикальному масштабуванню. Надає більш прості способи горизонтального масштабування. Це підвищує пропускну здатність і забезпечує стійку продуктивність майже в необмежених масштабах.
Надійність	Більш надійна	Менш надійна
Збереження та доступ до складних структур даних	Мають перевагу, оскільки були створені для роботи зі складними структурами даних	Можуть виникнути складнощі при роботі з дуже складними структурами даних
Продуктивність	Продуктивність головним чином залежить від дискової підсистеми. Для забезпечення максимальної продуктивності часто потрібна оптимізація запитів, індексів і структури таблиці.	Продуктивність зазвичай залежить від розміру кластера базового апаратного забезпечення, затримки мережі і викликаючого застосунку.

4.5.1 Firebase Realtime Database

Кожна СУБД має свої переваги та недоліки. Аналіз майбутнього функціоналу застосунку показав, що для реалізації запланованих можливостей зручним буде Firebase Realtime Database [5]. Це база даних — NoSQL із підтримкою SDK для iOS, Android і web. Вона легко інтегрується з іншими інструментами Firebase для аутентифікації, зберігання файлів, аналітики та інших. Цей інструмент зберігає дані в документах JSON. Синхронізація даних використовує веб-сокети, що дозволяють дуже швидко виконувати транзакції. База даних Realtime також обробляє оновлення, коли пристрій перебуває в автономному режимі, синхронізуючи зміни для користувача під час повторного підключення мережі. Розробникам подобається, як швидко вони можуть налаштувати серверну базу даних в реальному часі, і не потрібно турбуватися про такі події, як розгортання, апаратне забезпечення, час роботи та масштабованість. Ви можете мати досить надійний сервер для роботи за кілька хвилин. База даних реального часу вимагає, щоб розробники писали більшу частину коду програми на клієнті. База даних Realtime не має понять типів даних. Це дозволяє зберігати значення як рядки або числа, або вкладені об'єкти і масиви рядків і чисел у будь-яке поле. Масиви, однак, дійсно є просто об'єктами з індексами, що використовуються як ключі. Розробник самостійно керує цими типами даних. Запитує і реагує в режимі реального часу за допомогою бази даних Realtime, і це чудово працює. При необхідності встановити обмеження — встановлюються правила безпеки.

Існують наступні правила безпеки:

- — read (визначає хто може читати дані з бази даних);
- — write (визначає хто може записувати дані в базу даних);
- validate (визначає якого типу даних належить поле);
- indexOn (використовується для більш швидкого пошуку даних, шляхом виключення вузлів з пошуку);

Однією з чудових функцій баз даних Realtime Database є простий спосіб імпорту та експорту даних (через прості файли JSON) у консолі Firebase. Ця функція корисна при перенесенні даних, наприклад, з середовища розробки в середовище налаштування. Це також може бути корисним інструментом, коли не-розробникам необхідно швидко додавати або редагувати дані в базі даних; немає необхідності в скриптах або API, лише кілька кліків кнопки і можна завантажити повністю нові набори даних для програми.

Критерії, за якими було обрано Firebase Realtime Database:

- система може легко відстежувати кілька потоків даних в реальному часі;
- розраховується не надто багато невеликих записів / зчитувань з бази даних;
- зручна робота з JSON;
- відсутність складних структур даних;

При розробці даного програмного продукту було прийнято рішення виконувати усі маніпуляції з даними, такі як: фільтрація, пошук, сортування та інші, на стороні клієнта, оскільки це може значно пришвидшити роботу завдяки зменшенню кількості запитів і відсутності затримки, що може бути спровокована очікуванням відповіді від сервера. У розроблюваному застосунку такий підхід стає можливим, по-перше завдяки тому, що сучасні пристрої мають достатньо високі показники продуктивності, а, по-друге, застосунок не передбачає дуже складних маніпуляцій і обробки надто великого об'єму даних. Такий підхід також зручний при нестабільному підключенні до інтернету, оскільки для використання певних функціональних можливостей воно не є обов'язковим.

4.6 Реалізація бази даних розроблюваного застосунку

Для роботи з Firebase Realtime Database необхідно підключити застосунок до Firebase за допомогою конфігурації облікового запису, який спершу необхідно

зареєструвати і створити в ньому проект для подальшої роботи. Підключення бази даних проходить шляхом ініціалізації конфігурації облікового запису, що наводиться на рисунку 4.5. Для ініціалізації облікового запису використовується метод initializeApp(), котрий приймає конфігураційний об’єкт. У цьому об’єкті вказуються шляхи до проекту, ідентифікатор проекту у Firebase, авторизаційний домен та apiKey, який розробник отримує при реєстрації.

```
var firebaseConfig = {
  apiKey: "AIzaSyAxT3zizxtTPrD3Srw1VeLH1_WdBYbLX98",
  authDomain: "diplomaproject-ee019.firebaseapp.com",
  databaseURL: "https://diplomaproject-ee019.firebaseio.com",
  projectId: "diplomaproject-ee019",
  storageBucket: "diplomaproject-ee019.appspot.com",
  messagingSenderId: "23167646503",
  appId: "1:23167646503:web:15bbfa1c98d8eb7d"
};

firebase.initializeApp(firebaseConfig);
const storage = firebase.storage();
const database = firebase.database();
export { storage, database, firebase as default };
```

Рисунок 4.5 — лістинг підключення бази даних

На рисунку 4.6 наведено лістинг коду, який робить запис до бази даних. За допомогою методу «ref()» обирається, якщо був раніше створений, або створюється, якщо ні, підвузел до вузла «notes», ключ якого співпадає з унікальним ідеетифікатором запису. Даний ключ генерується при створенні запису, або береться зі стану компонента якщо це вже створений раніше запис. Для додавання запису використовується метод «set()» Аналогічно відбувається оновлення та видалення даних, використовуючи методи «update()» та «remove()» відповідно.

```
let userRef = database.ref(`notes/${note_id}`);
userRef.set({
  note_id,
  note_content: editorHTMLContent,
  note_title,
  note_description,
  note_tag,
  note_color,
  note_date
});
```

Рисунок 4.6 — запис даних до бд

На прикладі цієї ж підпрограми, збереження текстової інформації, розглянемо зчитування даних з бд, рисунок 4.7. Аналогічно запису відбувається підключення до вузлу. За допомогою методу «once» встановлюється слухач виду «дані», що повертає дані про записи, що записуються для подальших дій у стан компоненту.

```
const notesRef = database.ref("notes");
   notesRef.once("value").then(snapshot => {
     return this.setState({
       items: Object.values(snapshot.val()),
       searchItems: Object.values(snapshot.val())
     });
   });
```

Рисунок 4.7 — отримання даних з бд

База даних містить в собі багато вузлів, які в свою чергу мають підвузли. Firebase Database надає можливість розробнику переглядати структуру документів бази даних у спеціальній консолі. Це є досить зручно, оскільки при розробці можна повністю контролювати процес обміну даними з базою. Як приклад на рисунку 4.8 наведена структура підвузла «notes», котрий зберігає записи користувача взята із консолі.

Рисунок 4.8 — Структура вузлів Firebase Database на прикладі вузла «notes»

4.7 Структура проекту

Кожен проект має певну структуру папок та файлів. Для реалізації дипломного проекту було обрано один з популярних підходів організації скелету застосунку [18]. На Рисунку 4.9 наведено загальну структуру проекту. Усі файли проекту пов’язуються між собою завдяки використанню інструменту для збірки модулів Webpack.

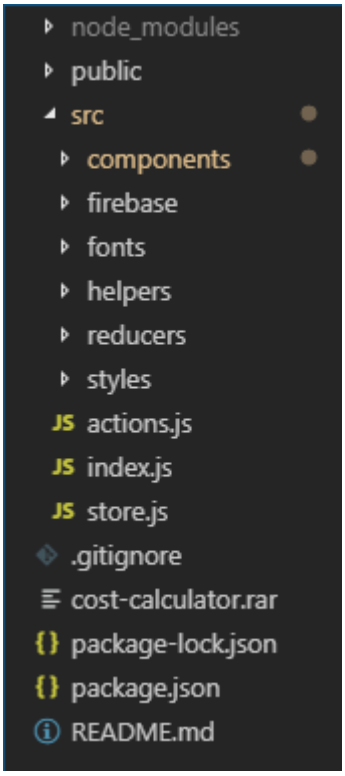

Рисунок 4.9 Структура проекту

Папка components містить підпапки, кожна з яких відповідає окремому компоненту. У кожній з цих підпапок лежить файл що має назву компонента і розширення .js, в якому лежить увесь код компоненту, а також на рівні з ним лежить файл стилів компоненту та index.js файл у який імпортується компонент, для того щоб спрощувати імпорт компонентів всередину обгортуючих компонентів. Такий підхід дозволяє позбавитися дублювання назви директорії та назви компоненту у шляхах при підключенні цього компоненту, оскільки якщо не вказувати назву компонента при імпорті, а тільки його директорію, то за замовчуванням імпортується файл, що називається index.js. Ця дрібниця робить код більш чистим та професійним. Те що кожен компонент знаходиться в окремій папці робить роботу з ним набагато зручнішою, адже не потрібно шукати потрібні файли стилів та логікі серед безлічі інших. Все зручно згруповано і перевикористання компонентів стає зручним. Папка firebase містить файли налагодження і підключення бази даних firebase database. У папці fonts містяться усі шрифти, використовуються на проекті. Папка helpers містить у собі файли з

корисними функціями, які використовуються у різних компонентах і мають загальне призначення. Також є директорія reducers, в якій лежать згадані раніше «редьюсери» з бібліотеки redux. Також стосовно Redux, є файл actions.js, в якому створюються так звані «action creators», що використовуються при створенні дії. Ну і також можна бачити папку styles, де лежать стилі, загальні для всього проекту. Файл Index.js лежить у корені застосунку вставляє усю побудовану компонентну структуру у DOM дерево.

При побудові застосунку з використанням React.js були дотриман усіх важливих правил і принципів. Одним з головних правил є розділення логіки і візуальної частини на два окремі компоненти. Це значно покращує читабельність коду, а також дозволяє ізолювати відображення компоненту від даних, що мають відрендеритися, тобто якщо для якогось одного компоненту потрібний такий же вигляд, що і для іншого, а логіка і дані відрізняються, можна без усілякого копіювання використати один і той же компонент, шо описує відображення просто імпортувавши його в компонент обгортку і передавши йому усі необхідні для рендерингу дані та методи, через властивості. Взагалі існує два типи компонентів. Це компонент-функція та компонент-клас. Компонент-клас може мати свій стан, може керуватися, Redux, а також має функції життєвого циклу. Реалізовуючи програмний продукт, я намагався використовувати якогомога більше функціональних компонентів, і менше компонентів-класів, адже такий підхід значно спрощує тестування застосунку і ідентифікацію можливих помилок. Немає нічого простішого ніж тестування чистих функцій. Чиста функція, це така функція, що при однакових аргументах завжди повертає однакові значення і її вивід залежить тільки від отриманого аргументу. Чиста функція не має побічних ефектів, тобто не змінює нічого в системі, що буде помітно для зовнішніх елементів програми.

У процесі роботи кожен компонент проходить через ряд етапів життєвого циклу. На кожному з етапів викликається той чи інший метод життєвого циклу. В

арсеналі React існує декілька методів життєвого циклу [3, 12], які активно використовуються при реалізації того чи іншого функціоналу, серед них:

- componentWillMount(): викликається безпосередньо перед відображенням DOM і використовується для ініціалізації бібліотек, створених іншими розробниками. Запуску анімацій, запиту даних чи виконання будь-яких додаткових налаштувань що можуть знадобитися перед відображенням компоненту на екрані;
- render(): рендеринг компоненту;
- componentDidMount(): являється ще одним місцем, звідки зручно відправляти запити до API. Ві викликається після відображення компонету на екрані, тому будь-які виклики методу для зміни стану компоненту призводять до запуску життєвого циклу оновлення і до повторного відображення компоненту на екрані. Також підходить для ініціалізації будь-якого коду JavaScript, що створений сторонніми розробниками, якому потрібен DOM;
- componentWillUnmount: викликається перед видаленням компоненту з DOM, може бути використаний для зупинки будь-яких процесів, що запущені в componentDidMount та componentWillMount;
- shouldComponentUpdate(nextProps, nextState): викликається кожен раз при оновленны об`єктів властивостей та станів. У якості параметрів передаються нові об’єкти властвостей та станів. Ця функція повертає true або false, від чого залежить виконання наступних функцій;
- componentWillUpdate(nextProps, nextState): викликається перед оновленням компоненту, якщо shouldComponentUpdate повертає true
- componentDidUpdate(prevProps, prevState): викликається одразу після оновлення компоненту;

Усі ці методи дозволяють контролювати стан компоненту на кожному кроці його зміни, і надають зручні способи управління ним.

4.8 Реалізація підпрограм

Проект реалізовано таким чином, що у його структурі існує тільки один єдиний HTML документ. У цьому документі один блок з ідентифікатором «root», в який вставляється уся DOM структура за допомогою методу render() з пакету ReactDOM. ReactDOM це пакет, що надає методи, специфічні для DOM, які використовуються на верхньому шарі розроблюваної програми.

Головним файлом, що формує ядро усього застосунку, є кореневий index.js. Якраз у цьому файлі відбувається вставка структури в HTML. Тут міститься головний компонент застосунку, що традиційно називається «App». Цей компонент обгортається такими компонентами як:

- Provider;
- ErrorBoundry;
- Router;

Кожен з них виконує окрему функцію, яка працює глобально, по всіх компонентах. Компонент Provider — це службовий компонент, імпортований з бібліотеки React-Redux, що приймає створене сховище, і надає доступ до нього іншим компонентам. Компонент Router це складова частинка пакету «react-router-dom». Цей пакет використовується для маршрутизації у React. Він зберігає інтерфейс застосунку синхронізованим з URL у браузері. React Router дозволяє маршрутизувати «поток даних» у застосунку зрозумілим способом. ErrorBoundry — це компонент-обгортка для відловлювання помилок застосунку.

На наступному рівні ієрархії компонентів застосунку, тобто всередині компоненту App підключається компонент MainPage, який містить посилання на решту компонентів, що є обгортками для компонентів кожної з підпрограм. А саме: Finances, Notes, TodoApp, WeatherForecast. Також у компоненті App підключено компоненти Login та Register, що відповідають за логування та реєстрацію відповідно. Реалізація кожної з підпрограм розглядатиметься в наступних підпунктах розділу.

4.8.1 Структурна схема застосунку

Кожен застосунок може мати декілька станів відображення, що представлені у вигляді різних сторінок. У React поділ на сторінки є умовним, оскільки це SPA, загальна схема інтерфейсу наведена на рисунку 4.10. Розроблюваний застосунок складається з наступних станів відображення:

– Екран авторизації
– Екран реєстрації
– Головна сторінка
– Початковий екран підпрограми фінансового контролю
– Екран підпрограми погоди
– Екран підпрограми заміток
– Екран підпрограми таск-менеджеру

Рисунок 4.10 — структурна схема застосунку

4.8.2 Реалізація підпрограми погоди

Підпрограма для відображення погодних умов базується на використанні стороннього API, що надає інформацію про погодні умови в певному регіоні, відповідно до запиту користувача. Для того щоб при першому завантаженні, користувач одразу бачив погоду у місці свого знаходження, використовується автоматичне визначення геолокації користувача. Для цього було використано звернення до об’єкту window.navigator, що містить інформацію про браузер, платформу та операційну систему користувача. Після визначення геолокації, що представляється у вигляді двох координат довготи та широти, до API посилається запит, що повертає об’єкт із великою кількістю інформації про погодні умови, відповідно до переданих у запиті get-параметрів.

Для отримання погодних умов при введенні користувачем назви населеного пункту, спершу необхідно отримати показники довготи та широти на основі введеної користувачем назви міста. Для цього також використовується сторонній API LocationIQ. Це зручний досить зручний та безкоштовний інструмент для двусторонньої конвертації географічних координат.

Уся робота з API супроводжується асинхронними запитами [15]. Для того щоб запобігти помилками активно використовувалися найсучасніші можливості JavaScript, такі як Promise та Async/await [13, 14]. Взагалі для асинхронних запитів в даному застосунку використовується вбудований у браузери Fetch API [19]. Це досить зручний інструмент, що заміняє звичний XMLHttpRequest, але на відміну від нього Fetch API побудований на Promises, які дозволяють використовувати більш простий та чистий API та запобігти катастрофічної кількості callback викликів, так званого “callback hell”.Це досягається завдяки можливості Promises створювати ланцюжки, що в свою чергу допомагає послідовно обробляти отримані дані.

Для зручної роботи зі сторонніми ресурсами було створено окремий клас, withWeatherService, що слугує для відправки запитів до сторонніх API та

отримання їх результатів. Для того щоб методи цього класу було легко використовувати у будь-якому компоненті, використовується Context API. Взагалі Context API виконує функцію глобальної передачі станів до усіх компонетів, але в даному випадку він використовується як зручний спосіб доступу до методів створеного сервісу, який не потребує зайвих імпортів. Для реалізації його було усі компоненти було обгорнуто у компонент DarkSkyServiceProvider, і кожен компонент, що потребує ці методи обгортається у компонент DarkSkyServiceConsumer. Перший слугує передавачем, а другий приймачем. Таким чином усі методи доступні через властивості компонента.

Для відображення діаграми погодних умов, було використано бібліотеку FushionCharts, яка має великий вибір різноманітних налаштувань для візуалізації даних в залежності від необхідного вигляду. У розроблюваному застосунку дані відображено у вигляді двох ламаних, що показують максимальний та мінімальний показники температур на декілька днів. Використана бібліотека вимагає строгу структуру об’єкту з параметрами для відображення. Тому для обробки даних що приходять із API DarkSky було створено конструктор WeekDataConstructor, що містить в собі набір методів для приведення об’єкту до необхідного вигляду. Цей конструктор імпортується до компоненту WeekChart і використовується у методі getWeatherData що викликається при завантаженні компоненту. При виклику конструктору аргументом йому передаються дані про погодні умови, що отримані від API. Також у методі getWeatherData формується конфігурація для побудови діаграми, яка передається через властивості до службового компоненту ReactFC, що безпосередньо відповідає за побудову діаграми.

4.8.3 Реалізація підпрограми фінансового контролю

На вершині ієрархії даної підпрограми лежать два головних компоненти, це ItemsPart та DetailsSection, які відповідно відповідають за відображення і створення фінансових операцій, та відображення детальної інформації про них.

Компонент ItemsPart складається з компоненту ItemCreator, що безпосередньо відповідає за створення фінансових операцій та компонентів CostList, FilterSection, Totals, які відповідають за відображення списку фінансових операцій, фільтрацію та підрахунок кінцевих результатів згідно існуючих фінансових операцій. За створення, а також редагування інформації операції відповідає компонент InputGroup, який в залежності від переданих властивостей виконує різні функції. Цей компонент має велику кількість значень у своєму стані, і якщо він використовується для створення нової операції, то усі ці значення пусті або дефолтні, а у випадку його використання для редагування, його стан встановлюють властивості, що беруться із масиву у сховищі, який складають усі існуючі записи, відповідно до ідентифікатору. Досить складно реалізовано фільтрацію записів, оскільки було поставлено задачу щоб усі фільтри працювали у зв’язці один з одним і при видаленні чи редагуванні запису за умов застосування фільтрів, має відображатися актуальний список із тими ж застосованими фільтрами враховуючи внесені зміни. Для цього було прийнято рішення створити у сховищі масив із записами, який би залежав від обраних фільтрів. Таким чином при внесенні глобальних змін до основного масиву записів, таких як редагування, додавання та видалення, усі компоненти оновлюються і одразу відображається актуальна інформація. Масив що містить список саме актуальних записів лежить у сховищі під назвою ItemsToShow і від нього відштовхуються решта компонентів, для відображення актуальної інформації. При цьому активно використовуються можливості згаданої раніше бібліотеки Redux для передачі усіх змін до сховища з одного компоненту і наступній їх передачі до решти компонентів, що мають перерендеритися відповідно до цих змін. В інструментах розробника Redux DevTool можна побачити діаграму стану сховища, що наведена на рисунку 4.11. Вона дозволяє глибоко дослідити дані, що лежать у сховищі. При розробці активно використовувалися дані можливості, оскільки взаємодія зі сховищем потребує постійного контролю та тестування.

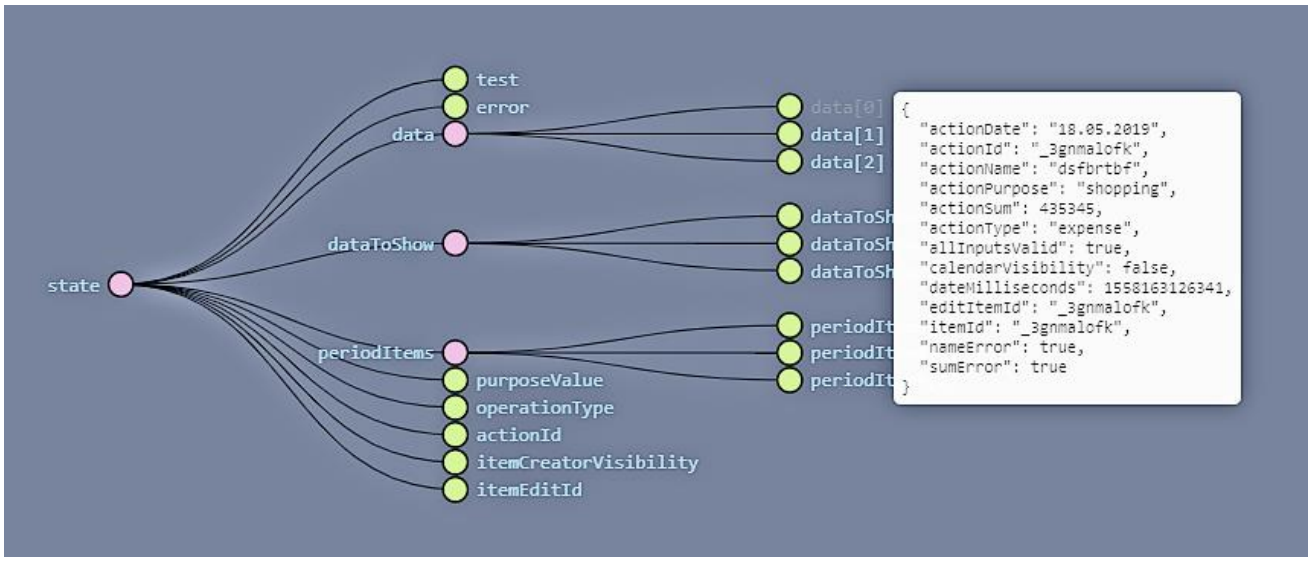

Рисунок 4.11 — структура сховища підпрограми фінансового контролю

На вершині ієрархії лежить об’єкт state який і є глобальним станом застосунку, і з яким відбувається взаємодія усіх компонентів. На наступних рівнях відображені поля, що лежать у state. Поле data являє собою масив усіх записів фінансових операцій і відповідно розгалуджується на окремі записи. Поле test це тестове поле, що було створене для перевірки роботоспроможності та правильного підключення Redux до розроблюваної програми. Поле error може містить значення false за замовчуванням і стає true, якщо при певній дії користувача відловлено помилку. При значенні цього поля true, застосунок відображає сповіщення про помилку. Наступне, вже згадане раніше поле dataToShow відповідає за відображення відфільтрованих та відсортованих даних, і залежить від поля data. Воно також являться масивом, що містить об’єкти записів фінансових операцій. Сам об’єкт фінансової операції складається з таких полей як:

– actionDate, відповідає за збереження дати
– actionID, ідентифікує запис
– actionName, заголовок запису
– actionPurpose, сфера, до якої належить запис
– actionSum, сума, вказана у записі

– actionType, тип фінансової операції
– allInputsValid, відповідає за валідацію при створенні запису
– calendarVisibility, вказує на стан календаря при створенні або редагуванні запису
– dateMilliseconds, дата створення запису

Також у глобальному стані програми наявні такі поля як purposeValue, operationType, actionId, itemCreatorVisibility, itemEditId, які відповідають за віддображення тих чи інших компонентів при різних діях користувача.

4.8.4 Реалізація підпрограми для зберігання текстових записів

В основі даної підпрограми лежить компонент Notes, що являється зовнішньою обгорткою для усіх компонентів. Цей компонент має свій локальний стан і у ньому згруповані методи, що пов’язують компоненти, які лежать у різних віхах ієрархії підпрограми. При написанні вимог до функціоналу, було вказано реалізація зручного текстового редактору, який дозволяв би редагувати текстові дані. Для вирішення цієї проблеми будо прийнято рішення застосувати готове рішення, що мало б увесь необхідний функціонал. Цим рішенням став пакет «react-draft-wysiwyg». Це рішення надає службовий компонент, який може мати свій стан, а також цей стан можна передавати через властивість editorState. При створенні нового запису цей стан спочатку пустий, і заповнюється по мірі написання тексту у редактор. А при редагуванні раніше створеного запису, через цю ж властивість до компонента передаються вже заповнені раніше дані, які піддаються зміні і редагуванню. Значення editorState має таку структуру, що не віддається подальшому використанню для відображення результуючого тексту, тому довелося користуватися такими пакетами для конвертації цієї структури як «draftjs-to-html» та «html-to-draftjs». Завдяки передачі компоненту тих чи інших службових властивостей можна легко керувати зовнішнім виглядом редактору. При створенні запису, одразу фіксується час його створення і відображається у

зручному форматі, у вигляді часу, що пройшов з моменту створення запису. Для вирішення цієї задачі, було використано спеціальний для цих цілей пакет, «react-time-ago», який у залежності від заданої маски відображає проміжок часу, від певного моменту, до теперішнього часу. Для цього йому достатньо передати властивість timeStyle, що відповідає за формат відображення часу та властивість date, яка приймає дату від якої починається відрахування часу, що пройшов.

4.8.5 Реалізація підпрограми таск-менеджеру

При реалізації даної підпрограми, її було обгорнуто у компонент TodoApp. Для того щоб мінімізувати кількість компонентів класів що мають власний стан, було вирішено підключити основні компоненти, що є певними вузлами застосунку до сховища і більшості дочірніх компонентів передавати дії та дані через властивості компонентів по ієрархії. А взаємодію з базою даних та сховищем підняти на якомога вищий рівень, оскільки чим більше компонентів-класів ти важчий процес розробки та тестування застосунку. Даний підхід має назву «Property drill». Усі дані при ініціалізації головного компоненту отримуються із бази даних і записуються до сховища. Потім компоненти-обгортки, що потребують ці дані, підключаються за допомогою компоненту Connect до сховища і отримують ці дані у своїх властивостях. Для того щоб дані відображалися відповідно до обраних фільтрів, використовуються допоміжні функції, що фільтрують їх відповідно до переданих значень і записують результат до локального стану, після чого компонент рендериться, використовуючи дані із локального стану. При будь-яких маніпуляціях із даними, що стосуються їх зміни, додавання або видалення головний компонент перерендерюється і отримує видозмінені дані з бази. Таким чином змінюється глобальний стан додатку, і усі компоненти, що підключені до сховища і використовують ці дані при відображенні також перерендерюються. Для спрощення отримання компонентами станів фільтрації, при усі фільтри записуються свій стан до

сховища, тому у компонентах, що підключені до сховища можна легко дістати із властивостей усі ці стани і передати їх дочірнім компонентам.

Висновки до розділу 4

У даному розділі були розглянуті основні архітектурні підходи та рішення, застосовані при розробці застосунку, проведено аналіз різних типів систем управління базами даних. На основі цього аналізу описано обгрунтування вибору СУБД, а також описано реалізацію бази даних. Також у підрозділах даного розділу описується побудова структури проекту, реалізація основних частин та функціональних можливостей розроблюваного застосунку.

5 ТЕСТУВАННЯ

5.1 Програмне забезпечення для реалізації тестування

Mocha (Mocha) — багатофункціональний тестовий JavaScript фреймворк, який працює на Node.js і в браузері, з підтримкою зручного асинхронного тестування [9]. Тести Mocha забезпечують точну і гнучку звітність і виконуються послідовно, обробляючи не перехоплені винятки. Цей фреймворк є потужним і одним з кращих інструментів для тестування.

Основні особливості фреймворку Mocha:

- підтримка тестування в браузері;
- рядкова diff (утиліта, що показує різницю між двома файлами);
- підтримка JavaScript API для запуску тестів;
- власний код на виході для підтримки безперервної інтеграції;
- відображення перехоплених винятків для правильних результатів тесту;
- асинхронна підтримка тайм-ауту тесту;
- тест конкретних тайм-аутів;
- підтримка повідомлень;
- маркування повільних тестів;
- авто-вихід (для запобігання "зависання" активного циклу роботи програми);
- проста мета-генерація тестових наборів і випадків;
- інтерактивні заголовки наборів для фільтрації виконання тесту;
- підтримка декількох викликів функції done ();
- довільна підтримка спрощеної мови (CoffeeScript і тд.);

5.2 Можливості фреймворку Mocha

Так як обраний фреймворк побудований на базі мови JavaScript, то він дозволяє проводити асинхронне тестування. Асинхронні тести на Mocha полягають в тому, що в функцію зворотного виклику передається додатковий параметр «done», який є функцією і викликається в разі успішного проходження тесту. Приклад створення завдання асинхронного наведено на рисунку 5.1

```
describe('User'), function () {
    describe('#save()'), function() {
        it('should save without error'), function(done) {
            var user = new User('Artem');
            user.save(done)
        });
    });
});
```

Рисунок 5.1 — приклад асинхронного тестування

Всі функції-хуки (before (), after (), beforeEach (), afterEach ()), що забезпечують різні умови, також підтримують асинхронну модель поведінки. Завдяки додатковому параметру done забезпечується проста підтримка асинхронного тестування.

Крім асинхронного тестування, також підтримується і синхронне тестування коду. Воно здійснюється за допомогою запуску функції зворотного виклику, яка забезпечує перехід до наступного тесту, в разі успішного завершення попереднього тесту. Синхронне тестування виглядає також як і асинхронне. Різниця в тому, що в синхронному тестуванні відсутній оператор done () через його непотрібність.

Mocha підтримує стандартну JavaScript-функцію — console.log (). Ця функція забезпечує висновок додаткової інформації в консоль під час роботи скриптів тестування, дозволяючи тестувальнику спостерігати за ходом виконання окремих частин функцій тестування.

Mocha дозволяє запускати тестування в різних режимах. За замовчуванням Mocha запускає виконання тестів один раз, завершує свою роботу і знову чекає команди запуску. Для того щоб відстежувати зміни в коді і запускати тестування автоматично, необхідно використовувати наступну команду — "$ mocha -watch". Завдяки автоматичному режиму, розробник позбавляється необхідності запуску тестування після кожної внесеної зміни в коді.

Функції тестування дозволяють здійснювати тестування певних наборів даних, використовуючи функцію only(). На рисунку 5.2 наведено приклад коду, який дозволяє вибрати певні типи даних серед інших значень і протестувати їх окремо. Такий гнучкий підхід дозволяє проводити тестування окремих частин застосунків, не зачіпаючи інші дані

```
describe('Array', function() {
    describe.only('#indexOf()', function() {
        // ...
    });
});
```

Рисунок 5.2 — тестування окремих частин коду

Функція skip () дуже схожа за способом роботи на функцію only (). Функція skip () дозволяє пропускати окремі частини даних, без проведення над ними тестів. На рисунку 5.3 наведено код, який дозволяє пропустити тестування певних частин даних. Така можливість іноді необхідна, у випадках, якщо тестувальник не знає як повинні змінюватися деякі частини тестованих даних.

```
describe('Array', function() {
    describe.skip('#indexOf()', function() {
        // ...
    });
});
```

Рисунок 5.3 — Використання функції skip () для пропуску тестування даних

Так як Mocha використовує виклики функціональних виразів для визначення специфікацій тестів, то це дозволяє створювати тести досить просто. При створенні тестів досить використовувати простий синтаксис мови JavaScript для досягнення аналогічної функціональності параметрезованих тестів у різних частинах коду, призначеного для проведення тестування.

Mocha дозволяє розробнику вибирати стиль мови програмування, спеціалізованого для конкретної області застосування (DSL - Domain Specific Language). Завдяки DSL в Mocha можна використовувати техніки BDD (behavior-driven development) і TDD (test-driven development), що робить Mocha універсальним інструментом для управління розробкою тестів. TDD - це техніка розробки програмного забезпечення, яка ґрунтується на повторенні дуже коротких циклів розробки: спочатку пишеться тест, що покриває бажану зміну, потім пишеться код, який дозволить пройти тест, і під кінець проводиться рефакторинг (процес зміни внутрішньої структури програми для полегшення читаності програмного коду) нового коду до відповідних стандартів. Техніка BDD тестування заснована на техніці TDD тестування з деякими змінами, які дозволяють більш ефективно працювати тестувальників.

5.3 Основні методи тестування програмного забезпечення

Тестування програмного забезпечення — перевірка відповідності між реальним і очікуваним поведінкою програми, яка здійснюється на кінцевому наборі тестів, обраному певним чином. Одними з процесів супутніх тестування, є процеси валідації та верифікації. Валідація — це визначення відповідності розроблюваного програмного забезпечення потребам і очікуванням користувача, вимогам до системи. Верифікація — це процес оцінки системи або її компонентів з метою визначення чи задовольняють результати поточного етапу розробки умов, сформованим на початку цього етапу. Іншими словами, чи виконуються цілі, терміни, завдання по розробці проекту, визначені на початку поточної фази . Після

оцінки, починається більш глибокий розгляд частин ПЗ, для реалізації тестових випадків. Тестовий випадок - це сукупність кроків, конкретних умов і параметрів, які необхідні для перевірки реалізації тестируемой функції або її частини. Якщо в ході тестування знаходяться дефектні місця ПЗ, звані багами. Розробка тестів для програмного забезпечення — це складний і копіткий процес, тому в багатьох випадках тестування відбувається тільки для окремих частин ПЗ. Для визначення кількості тестів, яке необхідно для перевірки окремої функції застосовується розрахунок тестового покриття, який проводиться за формулою: відношення кількості рядків коду, покритих тестами, до загальної кількості рядків коду, що тестується функції, помножене на 100%. Далі розглянемо основні методи тестування. Спадне тестування - метод, при якому програма тестується і збирається «зверху вниз». Окремо тестується тільки головний модуль. Після чого один за іншим з ним з'єднуються модулі, безпосередньо викликані їм, і тестується отримана комбінація. Так до тих пір, поки не будуть зібрані і перевірені всі модулі. Якщо викликаний для тестування модуль ще не існує, то для імітації функцій відсутніх модулів програмуються модулі «заглушки». Висхідне тестування — метод, при якому програма тестується і збирається «знизу вгору». Тільки модулі самого нижнього рівня тестуються ізольовано, автономно. Потім тестуються модулі, які безпосередньо викликають їх, які тестуються не автономно, а разом з уже перевіреними модулями. І так, поки не буде досягнута вершина. В останню чергу тестується програмне забезпечення в цілому. Крім основних видів тестування існують ще ряд таких, які засновані на взаємодії з головними модулями тестування, хоча деякі з них можуть бути застосовані автономно. Модульне тестування (юніт-тестування) — тестується мінімально можливий для тестування компонент, наприклад, окрема функція або клас. Часто модульне тестування здійснюється розробниками ПЗ. Інтеграційне тестування - тестуються інтерфейси між компонентами, підсистемами або системами. При наявності резерву часу на даній стадії тестування ведеться інтеграційно, з поступовим підключенням наступних підсистем. Системне тестування — тестується

інтегрована система на її відповідність вимогам. Після розробки програми та написання тестів для функцій, класів і решти програмного коду, розробники переходять до тестування застосунку в реальних умовах експлуатації. Дане тестування ділиться на два етапи: альфа- і бета-тестування. Альфа-тестування — це імітація реальної роботи з системою штатними розробниками, або реальна робота з системою потенційними користувачами / замовником. Найчастіше альфа-тестування проводиться на ранній стадії розробки продукту, але в деяких випадках може застосовуватися для закінченого продукту в якості внутрішнього приймального тестування. Іноді альфа-тестування виконується під відладником або з використанням оточення, яке допомагає швидко виявляти знайдені помилки. Бета-тестування — в деяких випадках виконується поширення попередньою для деякої більшої групи осіб, завдання якого упевнитися, що продукт містить досить мало помилок. Іноді бета-тестування виконується для того, щоб отримати зворотній зв'язок про продукт від його майбутніх користувачів. Часто, для вільного / відкритого ПЗ, стадія альфа-тестування характеризує функціональне наповнення коду, а бета-тестування — стадію виправлення помилок. Як правило, при цьому, на кожному етапі розробки проміжні результати роботи доступні кінцевим користувачам. Крім перерахованих вище видів тестування, існують ще так звані тестування "білого ящика" і "чорного ящика". дані назви використовуються для визначення того чи має розробник тестів доступ до вихідного коду тестованого ПЗ, або ж тестування виконується через інтерфейс або прикладний програмний інтерфейс, наданий тестованим модулем. При тестуванні білого ящика, розробник тесту має доступ до вихідного коду програм і може писати код, який пов'язаний з бібліотеками тестованого ПЗ. Це типово для юніт-тестування, при якому тестуються лише окремі частини системи. Воно забезпечує те, що компоненти конструкції — працездатні і стійкі до певної міри. При тестуванні білого ящика використовується мутаційне тестування — метод тестування програмного забезпечення, який полягає у внесенні невеликих змін до початкового коду програми. Відсутність помилок і невірних результатів при

тестуванні зміненої програми на наборі тестів може означати низьку якість наданого набору тестів або ж низьку значимість зміненої частини коду. Мутаційне тестування проводиться шляхом вибору мутаційних операторів і застосування їх одного за іншим до кожного фрагменту вихідного коду програми. Результат одного застосування мутаційного оператора до програми називається мутантом. Якщо набір тестів здатний виявити зміни (тобто один з тестів не проходить), то мутант називається убитим. При тестуванні чорного ящика у тестувальника є доступ до ПЗ виключно через ті ж інтерфейси, що і у замовника або користувача, або через зовнішні інтерфейси, що дозволяють іншому комп'ютера або іншому процесу підключитися до системи для тестування. Зазвичай, тестування чорного ящика ведеться з використанням специфікацій або інших документів, в яких описані вимоги до системи. Критерій покриття в даному виді тестування, як правило, складається з покриття структури вхідних даних, покриття вимог і покриття моделі (в тестуванні на основі моделей). При тестуванні сірого ящика розробник тесту має доступ до вихідного коду, але при безпосередньому виконанні тестів доступ до коду, як правило, не потрібен. Бета-тестування в цілому обмежене методом тестування чорного ящика.

5.4 Результати тестування

Тестування методом чорного ящика дозволило виявити багато дефектів ще у процесі розробки застосунку. Завдяки цьому було виявлено деякі баги зі збереженням записів у підпрограмі записнику, некоректне відображення результатів фільтрації операцій у підпрограмі фінансового контролю, діаграма, що відображає відсоток прибутків або витрат не змінювала свій вигляд при перемиканні фільтрів, оскільки не сприймала оновлення даних. Також підпрограма погоди видавала помилку через затримку відповіді від API.

В результаті проведення модульних тестів окремих компонентів, у компоненті Totals підпрограми фінансового контролю було виявлено некоректний

підрахунок результуючих сум доходів, витрат та поточного балансу. Решта тестованих компонентів пройшли тести успішно. Покриття модульними тестами досить кропіткий процес, через це багато компонентів залишилися невідтестованими, тому при продовженні розробки і підтримки даного застосунку необхідно покрити тестами більшість компонентів, що являються компонентами-класами, тобто мають свій стан.

Крім того було проведено тестування змін стану сховища за допомогою розширення браузеру Google Chrome Redux DevTools, рисунок 5.4. Але більшість помилок, що стосувалися роботи сховища як правило виявляються у процесі розробки. Тому помилок виявлено не було.

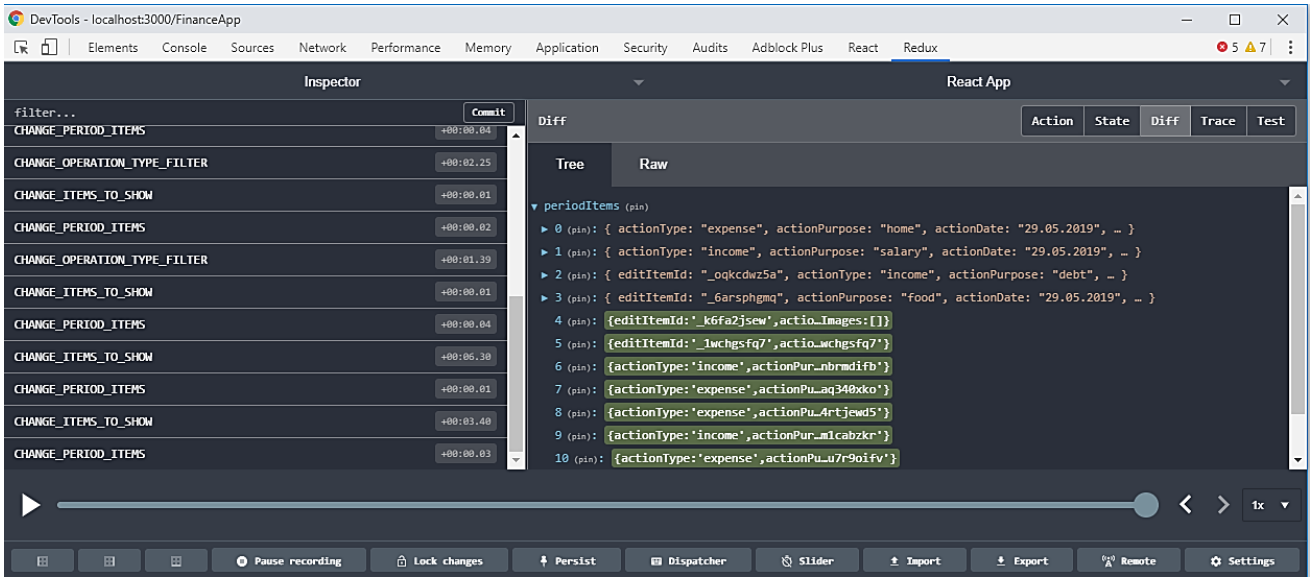

Рисунок 5.4 — вікно розширення Redux DevTools

Висновки до розділу 5

У даному розділі було розглянуто фреймворк для тестування Mocha, його основні можливості та переваги. Були наведені приклади практичного застосування даного інструменту, а також було роглянуто основні етапи та методи тестування, при розробці застосунків. В результаті тестування були виявлені і одразу ж виправлені помилки та баги.

6 ІНСТРУКЦІЯ КОРИСТУВАЧА

При відкритті застосунку, якщо користувач не був авторизований раніше, він потрапляє на екран авторизації. На ньому розташовано два поля для введення адреси електронної пошти та паролю, рисунок 6.1. У разі введення некоректних даних, або не співпадіння введених даних з існуючими у базі, користувач буде повідомлений про неможливість авторизації. У випадку успішної авторизації він потрапляє на головну сторінку. Для незареєстрованих користувачів надається можливість перейти на сторінку реєстрації. На екрані реєстрації розташовані стандартний набір полів для проведення реєстрації. При проходженні авторизації або ж реєстрації, користувач направляється на головну сторінку.

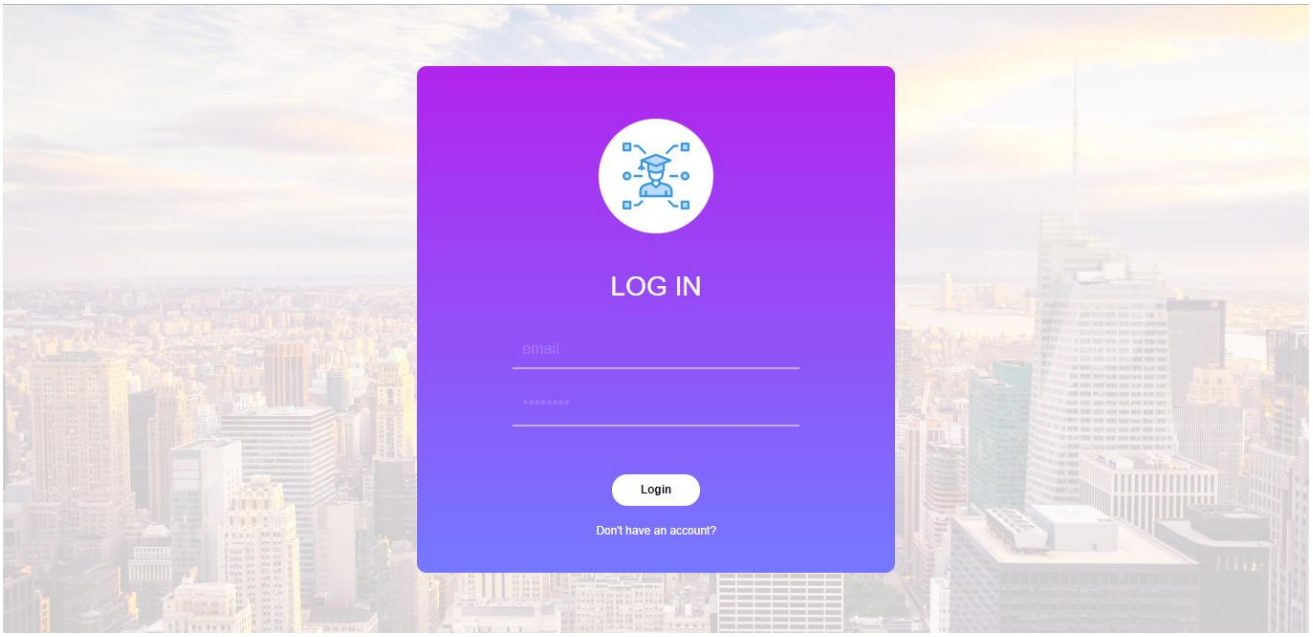

Рисунок 6.1 — екран авторизації

Основною задачею головної сторінки є перенаправлення користувача до необхідної йому підпрограми. Тому на ньому відображені блоки, що містять посилання на ці підпрограми рисунок 6.2. Для переходу у будь-яку підпрограму користувач має клікнути по блоку, що відповідає обраній підпрограмі.

Рисунок 6.2 — головна сторінка застосунку

Для повернення з підпрограми до головної сторінки або переходу до іншої підпрограми, на кожному екрані присутній елемент, що має значок «бургеру» і розташовується у правому верхньому кутку екрану. При кліку на цей елемент з’являється спливаче меню, яке містить посилання для переходу.

6.1 Інструкція до підпрограми фінансового контролю

На початковому екрані даної підпрограми, з лівої сторони знаходиться область, що відповідає за відображення існуючих записів фінансових операцій, а з правого — певні статистичні дані щодо фінансових операцій, враховуючи діаграму. На рисунку 6.3 пронумеровані елементи, з якими може взаємодіяти корисутвач.

Для відкриття блоку, що відповідає за створення нового запису необхідно клікнути по кнопці, що позначена цифрою «1». У нижній частині екрану застосунку, розташовано елементи що відповідають за фільтрацію записів. Для фільтрації за датою необхідно клікнути в області, що позначена цифрою 2, після чого відкриється календар При кліку на область, що позначена цифрою «3»

відкривається блок з календарем, що дозволяє обрати початкову та кінцеву дати в межах яких необхідно відфільтрувати записи. Для закриття цього блоку необхідно обрати обидві дати або натиснути на елемент, що позначений цифрою «2». Для фільтрації записів за типом (надходження, витрати, всі) необхідно натиснути на один з перемикачів, позначених цифрою «4». Натиснувши на елемент, позначений цифрою «8» користувач може здійснити фільтрацію за певною категорією фінансових операцій, обравши у блоці що відкрився по натисненню певну категорію. Для того щоб відредагувати запис, користувачеві необхідно натиснути на значок з олівцем, позначений цифрою «5» в області, що відповідає саме тому запису, який він хоче змінити. Щоб видалити запис необхідно натиснути на значок сміттєвої корзини, позначений цифрою «6», аналогічно у області запису, що підлягає видаленню.

Рисунок 6.3 — початковий екран підпрограми фінансового контролю

6.2 Інструкція до підпрограми таск-менеджеру

На початковому екрані підпрограми відображається список завдань, що поставлені на поточну дату, з лівої сторони розташовуються фільтри, які

допомагають швидко переглянути необхідну інформацію. У верхному лівому кутку розташований фільтр у вигляді календаря, який дозволяє обрати дату, відповідно до якої необхідно переглянути список задач. Нижче відображено список груп завдань, відповідно обравши одну з них, відобразяться записи, що належать обраній групі. Ще нижче розташовується фільтрація записів по стану виконання, що включає в себе 3 стани: «усі», «в процесі виконання» та «виконані». У верхній частині екрану розташовується область пошуку, яка автоматично фільтрує записи або групи при введенні пошукового запиту. Справа від пошуку, знаходяться переключателі відображення списку завдань, та груп, відповідно переключивши які, в області що займає більшу частину екрану, відображаються елементи завдань чи груп. Для швидкого управління задачами у завданні, можна натиснути на будь-яку задачу і відповідно відмітити її як виконану або невиконану.

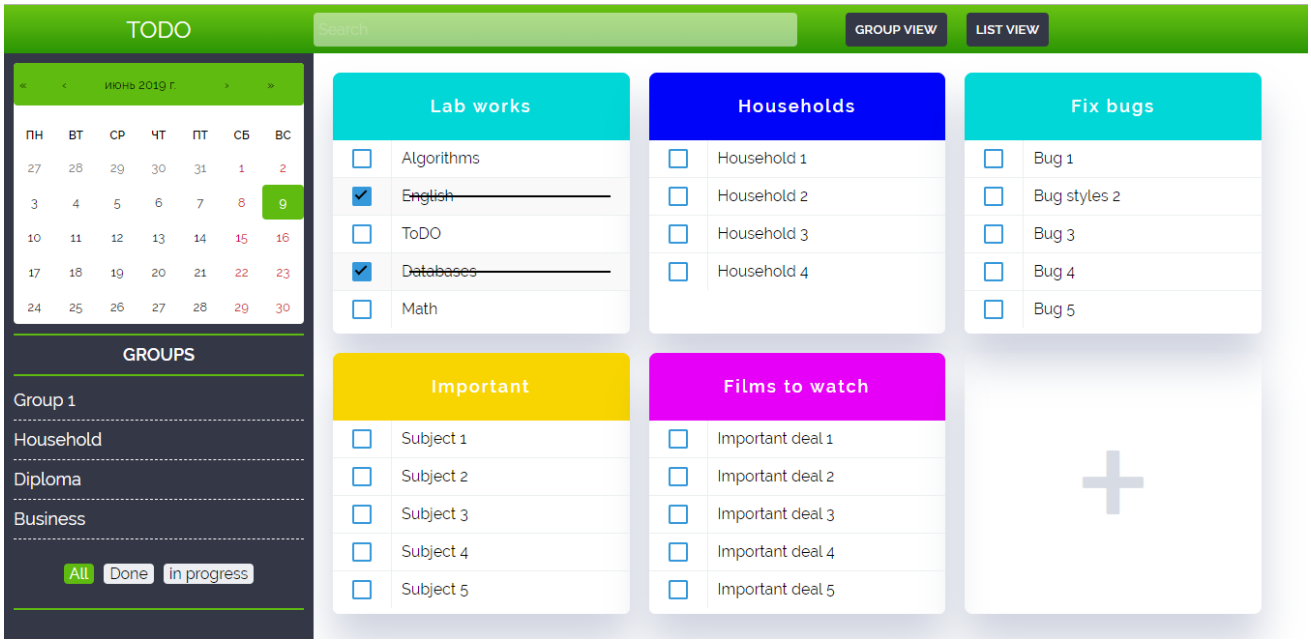

Рисунок 6.4 — початковий екран підпрограми таск-менеджеру

При натисканні на область назви завдання, відкривається екран докладного вигляду цього завдання, рисунок 6.5.

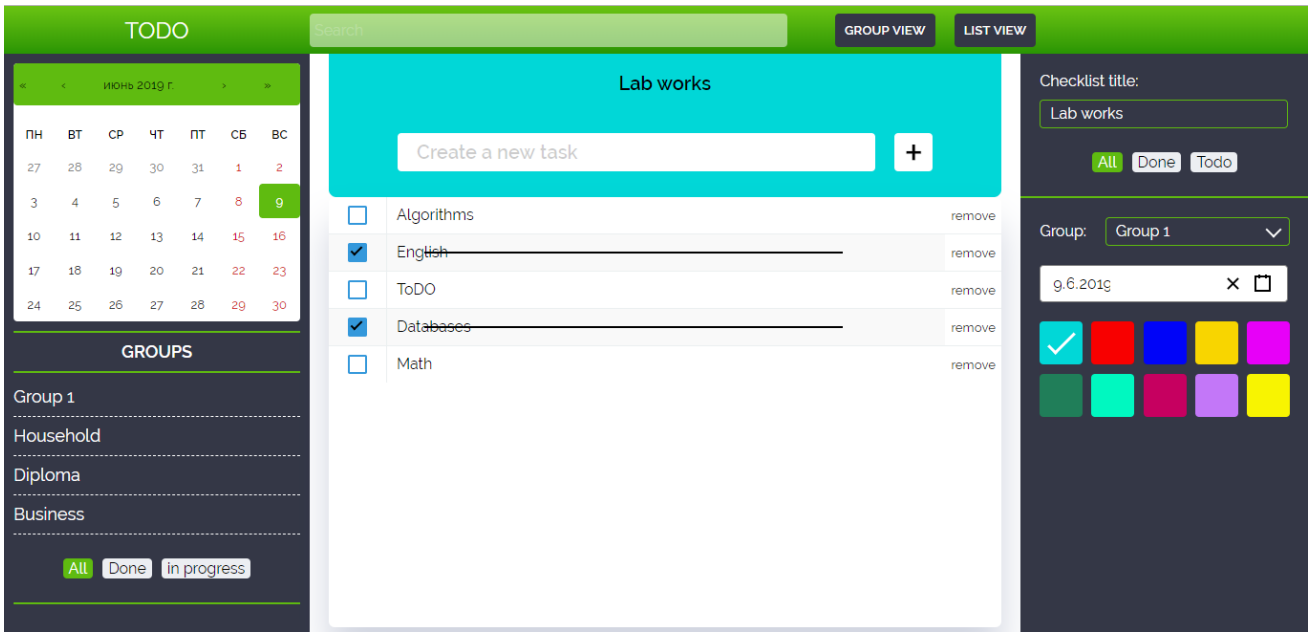

Рисунок 6.5 — Докладний вигляд завдання

На даному екрані представляється можливість створювати задачі у завданні. Для цього необхідно ввести текст задачі в поле, що знаходиться над списком задач і має підпис «Create a new task», після чого можна у праві стороні екрану можна обрати дату для завдання, групу та колір. Також можна відфільтрувати задачі в завданні по статусу їх виконання.

6.3 Інструкція до підпрограми записника

Дана підпрограма має тільки один екран, рисунок 6.6. У лівій частині знаходиться область , де розташовується список тегів, за якими можна відфільтрувати записи. Правіше від цієї області відображено область що містить пошук та список усіх записів. Для пошуку, достатньо ввести запит. На кожному записі можна бачити іконку кошику, натискання на яку видаляє запис. При натисканні на назву запису, в області що знаходиться правіше, відображається уся докладна інформація , що стосується цього запису. Якщо жоден запис не відкритий, ця область слугує для створення записів. В цій області розташовані такі поля для назви, короткого опису, а крім цих полів можливість обрати колір для

запису та створити тег. Усю праву частину екрану займає Текстовий редактор, що відповідає за відображення основного вмісту запису. У верхній частині цього блоку розташовано блок з кнопками, які відповідають за візуальне оформлення тексту. Для того щоб змінити вигляд тексту, потрібно виділити необхідну його частину і задати усі необхідні параметри оформлення, що доступні на панелі вгорі. Для підтвердження змін у записі, або створення нового, необхідно натиснути кнопку «Apply», що знаходиться у правому нижньому кутку.

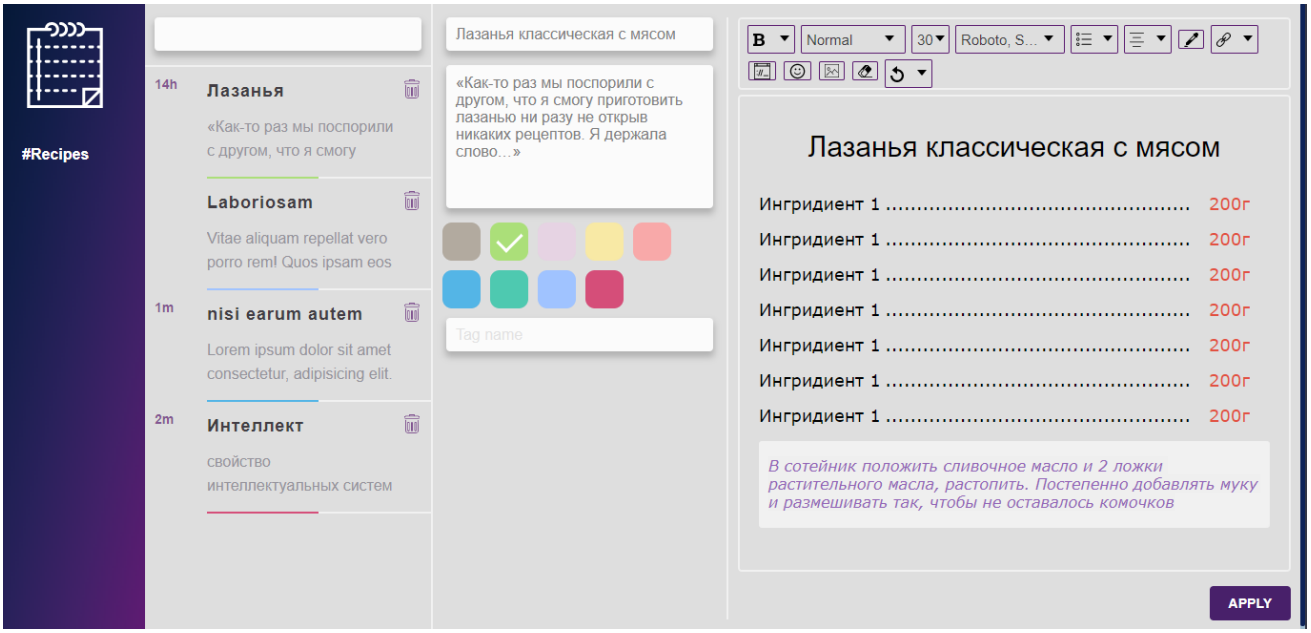

Рисунок 6.6 — екран підпрограми записника

Висновки до розділу 6

В даному розділі були описані основні можливості взаємодії з інтерфейсом застосунку. Надано докладну інструкцію щодо виконання тих чи інших завдань користувача. А також наведені приклади використання деяких можливостей кожної з підпрограм.

ВИСНОВКИ

У ході виконання дипломного проекту, було розроблено завершений веб-застосунок для допомоги людям в організації повсякденного життя. Були розглянуті існуючі програмні рішення, їх переваги та недоліки, які були враховані під час розробки. Кінцевий продукт має широкий спектр можливостей. Більшість поставлених на етапі планування вимог було реалізовано.

Використаний SDK Firebase виявився достатньо зручним інструментом для розробки не дуже складних додатків, але при розробці більш масштабних проектів можуть виникати складнощі із використанням Firebase Database, оскільки складна вибірка даних з нереляційної бази даних є досить проблемним питанням. Загалом, усі обрані технології та архітектурні підходи задовольнили вимоги під час розробки.

Подальший розвиток даного застосунку буде направлено на покращення якості, додання нових функцій, дозволяючих розширити можливості застосунку, а також більш детальне пропрацювання інтерфейсу.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. 30 необходимых практик для написания современного и эффективного HTML5 [Електронний ресурс] — режим доступу: https://medium.com/@stasonmars/30-необходимых-практик-для-написания-современного-и-эффективного-html5-79da9b3f245
2. Способы организации CSS-кода [Електронний ресурс] — режим доступу: https://habr.com/ru/post/256109/
3. React — a JavaScript library for building user interfaces [Електронний ресурс] — режим доступу: https://reactjs.org/
4. Виртуальный DOM и детали его реализации в React [Електронний ресурс] — режим доступу: https://ru.reactjs.org/docs/faq-internals.html
5. Firebase Realtime Database [Електронний ресурс] — режим доступу: https://firebase.google.com/docs/database/
6. An introduction to the Flux architectural pattern [Електронний ресурс] — режим доступу: https://www.freecodecamp.org/news/an-introduction-to-the-flux-architectural-pattern-674ea74775c9/
7. A Firebase in React Tutorial for Beginners [2019] [Електронний ресурс] — режим доступу: https://www.robinwieruch.de/complete-firebase-authentication-react-tutorial/
8. Javascript-фреймворки: тенденции 2019 года [Електронний ресурс] — режим доступу: https://habr.com/ru/company/plarium/blog/433926/
9. Mocha – the fun, simple, flexible JavaScript test framework [Електронний ресурс] — режим доступу: https://mochajs.org/
10. Redux —a predictable state container for JavaScript apps [Електронний ресурс] — режим доступу: https://redux.js.org/
11. Eloquent JavaScript, 3rd edition apps [Електронний ресурс] — режим доступу: https://eloquentjavascript.net/

12. Learning React Functional Web Development with React and Redux [текст] / Alex Banks and Eve Porcello, O'Reilly Media, 2017
13. Современный учебник JavaScript [Електронний ресурс] — режим доступу: https://learn.javascript.ru/
14. Promise [Електронний ресурс] — режим доступу: https://developer.mozilla.org/enUS/docs/Web/JavaScript/Reference/Global_Objects/Promise
15. Асинхронность в JavaScript: Пособие для тех, кто хочет разобраться [Електронний ресурс] — режим доступу: https://habr.com/ru/company/wrike/blog/302896/
16. Serverless Web Applications with React and Firebase: Develop real-time applications for web and mobile platforms [текст] / Harmit Singh, Mayur Tanna — Packt Publishing, 2018
17. How to use Redux in ReactJS with real-life examples [Електронний ресурс] — режим доступу: https://www.freecodecamp.org/news/how-to-use-redux-in-reactjs-with-real-life-examples-687ab4441b85/
18. React Architecture Best Practices [Електронний ресурс] — режим доступу: https://www.sitepoint.com/react-architecture-best-practices/
19. Fetch API [Електронний ресурс] — режим доступу: https://developer.mozilla.org/ru/docs/Web/API/Fetch_API